VŠB - Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Vývoj herních aplikací na iOS a Android OS

Games Development on iOS and Android OS

2012 Bc. Jakub Holoubek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jakub Holoubek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Vývoj herních aplikací na platformách iOS a Android OS
Games Development on iOS and Android OS

Zásady pro vypracování:

Cílem práce je vývoj a návrh herních aplikací určených pro mobilní zařízení s operačním systémem iOS a Android. Realizace prostřednictvím inkrementálně řízeného projektu. Návrh, analýza, design, rollout na prodejní platformu. Popis výhod/nevýhod jazyku LUA. Práce s 3D objekty v prostředí Shiva 3D a jejich interakce s herní logikou. Vytvoření jednoduchých fyzikálních modelů.
1. Analýza aplikace prostřednictvím objektové metodologie postavené na UML specifikace.
2. Práce s 3D objekty.
3. Implementace v jazyce LUA - v prostředí Shiva 3D (společnost StoneTrip).
4. Interakce 3D modelů z okolím a jejich jednoduché fyzikální modely.
5. Verzování aplikace, vytvoření vývojového prostředí pro práci v týmu.
6. Testování, návrh testovacích scénářů.
7. Výstupem práce bude hra kombinující prvky 3D akce a strategie.

Seznam doporučené odborné literatury:

James Steele, Nelson To, The Android Developer's Cookbook: Building Applications with the Android SDK, Addison-Wesley Professional, 2010, ISBN-13: 978-0321741233
Reto Meier, Professional Android 2 Application Development, Wrox, 2010, ISBN-13: 978-0470565520
Luiz Henrique de Figueiredo, Lua Programming Gems, Lua.org, 2008, ISBN 978-8590379843

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Tomáš Ivanský**

Konzultant diplomové práce: Ing. Michal Krumnikl

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava.

Použité diagramy, zdrojový kód a grafika jsou vlastnictvím společnosti Zoongo, s. r. o. a mohou být šířeny pouze s jejím souhlasem.

V Ostravě dne 27.4.2012 .....

podpis

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 27.4.2012 .....

podpis

PODĚKOVÁNÍ

Děkuji Mgr. Tomáši Ivanskému za vedení mé diplomové práce a umožnění spolupráce na projektu Re!Sources.

Děkuji celému týmu společnosti Zoongo, s. r. o. za spolupráci a rady při řešení některých problémů.

Děkuji Ing. Michalu Krumniklovi za konzultace spojené s mou diplomovou prací.

ABSTRAKT

Tato diplomová práce je napsána díky spolupráci s firmou Zoongo, s. r. o., ve které jsme pracovali na vývoji hry pro mobilní zařízení s operačními systémy iOS a Android OS, která využívá prvky trojrozměrného světa. V textové části bude popsána konečná podoba hry včetně jejího návrhu a implementace. K vývoji byl použit nástroj pro vývoj multiplatformních aplikací od společnosti Stonetrip – Shiva 3D. Všechny zdrojové kódy jsou v jazyce Lua, který je standardním jazykem pro vývoj v prostředí Shiva 3D. Hotová aplikace bude dále distribuována pomocí on-line obchodů App Store a Google Play.

KLÍČOVÁ SLOVA

Stonetrip Shiva 3D, Lua, Apple iOS, Google Android, mobilní zařízení, hra, iPhone, iPad

ABSTRACT

This thesis is written trough collaboration with Zoongo, s. r. o. company, in which we worked on a game development for mobile devices with operating systems iOS and Android OS, which use elements of three-dimensional world. The final version of the game, including design and implementation, is described in the text part of thesis. Tool for multiplatform applications called Shiva 3D from Stonetrip was used for the development. All source codes are written in Lua language, which is the standard language for development in Shiva 3D. The final application will be distributed through online markets App Store and Google Play.

KEY WORDS

Stonetrip Shiva 3D, Lua, Apple iOS, Google Android, mobile device, game, iPhone, iPad

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SDK	Software Development Kit
NDK	Native Development Kit
WYSIWYG	What You See Is What Yout Get
DWF	Design Web Format
VoIP	Voice Over Internet Protocol
FPS	Frames Per Second
TPS	Third Person Shooter
GPL	General Public License
SVN	Subversion
RAID	Redundant Array of Inexpensive/Independent Disks
OOP	Objektově orientované programování
UML	Unified Modeling Language

OBSAH

1. Úvod

Tématem mé diplomové práce byl vývoj herní aplikace pro zařízení s iOS[1] a Android OS[2]. Jedná se o externí téma, které jsem vypracovával pod společností Zoongo s. r. o. V týmu pěti lidí jsme vyvíjeli hru, kterou jsme úspěšně vydali a umístili na Apple App Store[3]. Každý z týmu měl na starosti pouze určitou část celého vývojového cyklu. Já jsem se především zabýval vývojem, který byl spojen s vytvářením a správou 3D objektů, které ve hře používáme.

Nejprve popíši nástroj, který jsme se pro implementaci rozhodli použít. Zaměřím se na některé jeho části a také na srovnání s konkurenčními nástroji.

Následující kapitoly budou spíše teoretické. Zmíním se v nich o programovacím jazyku, který jsme při implementaci používali, jazyku UML[4], který jsme využívali při návrhu jednotlivých částí aplikace. Díky tomu, že jsme hru vyvíjeli v týmu, bylo nutné použít nástroj, který by nám umožňoval správu verzí aplikace, proto se zde také krátce zmíním o systému SVN[5]. To je ve stručnosti vše, co se týká teoretické části.

Ve zbývající části se budu zaobírat samotným vývojem projektu, který jsme nazvali Re!Sources. Postupně zde popíši, o jaký typ hry se jedná, co je hráčovým cílem a jak se k němu může dostat. Dále se zaměřím na ty části hry, na kterých jsem pracoval převážně já. Vždy samotný problém popíši, poté se zmíním o navrhovaném řešení a u některých problémů také ukáži detaily toho, jak jsme se jej rozhodli implementovat.

Úplně na závěr pak v krátkosti popíši, jakým způsobem je možné vytvořit balíčky, které je možné instalovat do koncových zařízení s iOS a Android OS.

[1] http://www.apple.com/ios/
[2] http://www.android.com/
[3] http://store.apple.com/us
[4] http://www.uml.org/
[5] http://subversion.apache.org/

2. STONETRIP SHIVA 3D[6]

Shiva 3D je komplexní vývojový nástroj pro tvorbu aplikací a 3D her spustitelných na mnoha platformách. Tento nástroj je vyvíjen společností Stonetrip. Poslední verze byla vydána 12. prosince 2011 a nese označení Shiva 3D 1.9.1. Vyvíjet v tomto prostředí je nyní možné pouze na operačním systému Windows, pro distribuci na určitá cílová zařízení je však někdy nutné použít jiný operační systém.

V současné době Shiva podporuje export do těchto platforem:

- *Windows*
- *Mac OS*
- *Linux*
- *iOS*
- *Android*
- *HP WebOS*
- *Marmalade*
- *Wii*
- *Možný export do webové aplikace*

2.1. SHIVA 3D – SOUČÁSTI

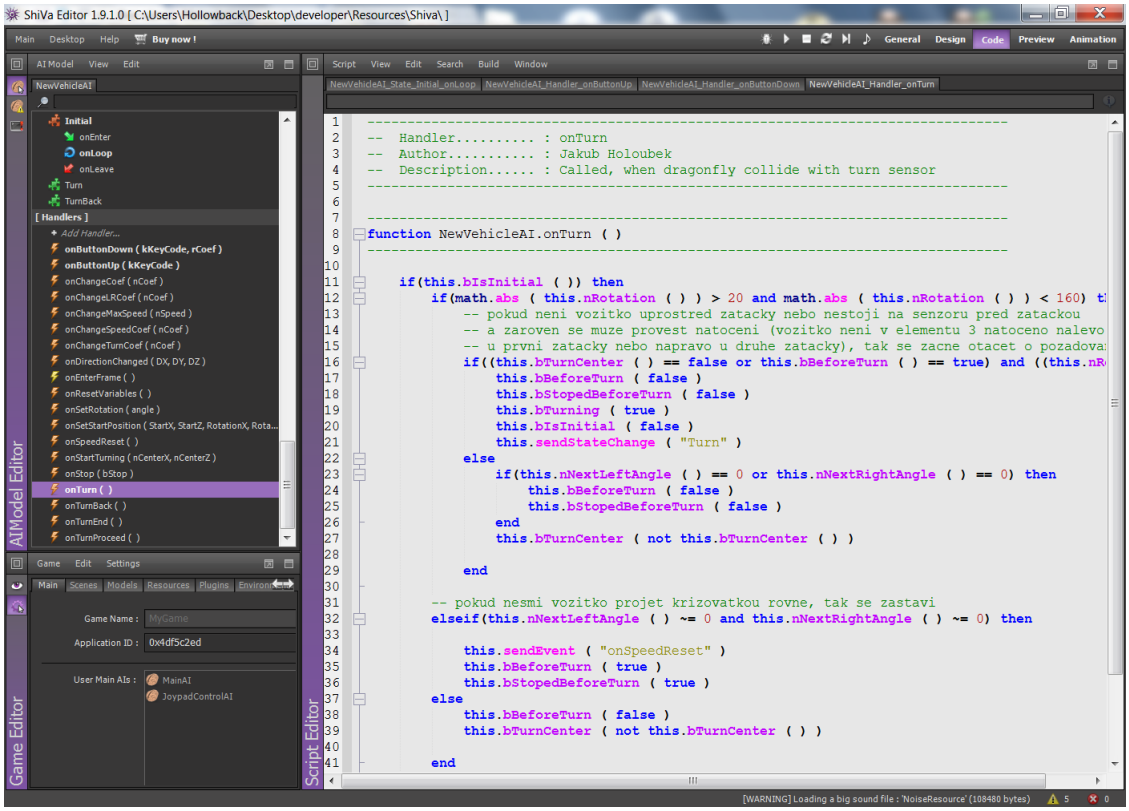
Před samotným nasazením aplikace na koncová zařízení je třeba projít „Shiva 3D vývojovým cyklem“. Nejprve je hra vyvíjena v Shiva 3D editoru, jehož výstup je použit jako vstup do nástroje Shiva 3D Authoring Tool. Tento nástroj pak přistupuje k jednotlivým SDK pro konkrétní zařízení a vygeneruje zdrojový kód pro cílovou platformu. Pro zařízení s iOS je vygenerován kód v Objective-C, pro Android OS zase v Javě.

Stonetrip nabízí hned několik nástrojů, jejichž použití je pro určité platformy více, či méně vyžadováno. Některé z těchto nástrojů lze dokonce používat jen na určitém operačním systému. Nyní si je postupně představíme.

[6] http://www.stonetrip.com/

2.1.1. SHIVA 3D EDITOR

První z nástrojů, které si popíšeme, je Shiva 3D Editor. Jedná se o jednu z hlavních součástí, bez které by vývoj Shiva 3D aplikací nebyl možný. Shiva 3D Editor je komplexní nástroj, který nám dává možnost vytvořit kompletní 3D aplikaci bez použití dalších podpůrných programů. Jak prostředí Shiva 3D editoru vypadá, můžete vidět na obrázku obr. 1.



obr. 1 – Shiva 3D editor

2.1.1.1. DATA EXPLORER

Nejprve si představíme funkci, díky které můžeme v aplikaci používat zdroje, jako jsou například 3D modely, textury, zvuky aj. Tuto funkčnost nám poskytuje modul „Data Explorer“. Je zde umožněn import 3D modelů ve formátu Collada nebo DWF, které podporuje většina programů pro tvorbu 3D grafiky. Dále můžeme importovat většinu hudební souborů, obrázků a také videí.

Abychom nebyli vázáni pouze na editory 3D grafiky třetích stran, je zde k dispozici funkce, díky které můžeme vytvářet základní 3D objekty jako krychli, kouli, válec aj.

2.1.1.2. SCENE VIEWER

Jedná se o modul, který nám umožňuje zobrazení scény aplikace zhruba tak, jak bude vypadat po vyrenderování. Jedná se vlastně o WYSIWYG editor pro 3D grafiku.

2.1.1.3. PARTICLE EDITOR

Každému objektu na scéně lze přiřadit tzv. systém částic. Díky tomu můžeme přímo ve hře vytvořit např. efekt kouře nebo ohně bez použití speciálních 3D nástrojů.

2.1.1.4. SCRIPT EDITOR

Nejpoužívanějším modulem je bezesporu „Script editor“. Jak již název napovídá, jedná se o modul, který slouží k práci se skripty. Oproti jiným nástrojům nenabízí nic nového. Pro usnadnění psaní kódu implementovali tvůrci do toho modulu intellisense. Prostředí tak vývojáři mnohdy napoví, když si nemůže vzpomenout na přesný název některé funkce/proměnné. V nové verzi Shiva 3D 1.9.1 byl přidán dlouho očekávaný debugger, který usnadňuje práci při hledání případné chyby.

2.1.1.5. ANIMATION

Editor v sobě obsahuje také modul pro základní práci s 3D animacemi. Složité animace na úrovni některých menších částí objektu s ním však provádět nelze. V tom se nemůže s velkými aplikacemi pro tvorbu 3D srovnávat. Pro animace typu otáčení tělesa dokola, pohyb po určité dráze apod. je to ovšem dostačující nástroj.

2.1.2. SHIVA 3D AUTHORING TOOL

Authoring Tool je důležitý nástroj, díky němuž jsme schopni publikovat aplikaci na různé typy koncových zařízení. Z STK souboru, který vygeneruje Shiva Editor, dokáže vygenerovat zdrojové soubory pro cílovou platformu jako je iOS, Android, Palm aj. K tomu, aby byl Authoring Tool schopen generovat kód pro cílovou platformu, je někdy zapotřebí mít nainstalovány SDK jednotlivých platforem.

2.1.3. SHIVA SERVER

Jedná se o aplikaci, která spravuje spojení mezi uživateli ve víceuživatelské aplikaci. V poslední verzi obsahuje také nástroje pro zprostředkování VoIP hovorů.

2.1.4. DEVICE DEVELOPMENT TOOLS

Shiva Editor má v sobě integrován nástroj, který umožňuje testovat vytvořené aplikace. Nevýhodou tohoto nástroje je, že aplikace dokáže reagovat pouze na vstupy z klávesnice nebo myši. Tuto nevýhodu se pokouší eliminovat nástroj Device Develpment Tools. Dovoluje nám totiž použít koncové zařízení jako iPhone, Palm nebo zařízení s Android OS jako gamepad

a simulovat v něm vstupy z akcelerometru, odchytávat vícenásobný dotyk na displej a jiné situace, které lze vyvolat pouze na koncovém zařízení. Jedná se tedy o aplikaci, která se pomocí Authoring toolu nahraje do koncového zařízení. Na počítači je poté potřeba zprovoznit server, na který jsou odesílána data z mobilního zařízení. K tomu, aby fungoval přenos dat, je potřeba zajistit, aby bylo zařízení a PC ve stejné síti.

2.2. SROVNÁNÍ S KONKURENCÍ

Shiva 3D není jediný engine pro vytváření multiplatformních 3D aplikací – je jich celá řada. Liší se v mnoha ohledech. Proto jsme před vývojem hledali ten nejvhodnější. V našem případě bylo hlavním požadavkem najít engine, který by podporoval distribuci výsledné aplikace na zařízení s iOS a Android OS. Tři nejvýznamnější jsem se rozhodl mezi sebou porovnat. Kromě námi zvoleného Shiva 3D jsem pro porovnání vybral Unity 3D[7] a SIO2[8].

	 SHIVA 3D	 UNITY 3D	 SIO2
Podporované platformy	Windows, Linux, Mac OS, iOS, Android, HP WebOS, BlackBerry QNX, Marmelade, Wii, webové prohlížeče	Windows, Mac OS, iOS, Android, Wii, PS3, Xbox 360, Flash, webové prohlížeče	Windows, Mac OS, iOS, Android
Programovací jazyk	Lua[9], C++	C#.NET, JavaScript, Boo	Lua, C++
Vývojové prostředí	Shiva 3D Editor	Unity Editor	XCode, Visual Studio, Eclipse
Cena	$ 400	cca $ 1000	$ 800

[7] http://unity3d.com/
[8] http://sio2interactive.com/
[9] http://www.lua.org/

3. LUA

Při vývoji aplikací využívajících Shiva 3D engine se téměř výhradně používá skriptovací jazyk Lua. Jedná se o vysokoúrovňový skriptovací jazyk jako jsou např. Perl, Python, JavaScript aj. Tento jazyk se stal svou syntaxi a dalšími prvky, kterými vývoj usnadňuje, u vývojářů her velmi oblíbený. Syntaxe některých programových konstrukcí je podobná programovacímu jazyku Pascal.

Jazyk jako samotný nemá přímou podporu pro objektově orientované programování. Toto lze sice různými způsoby obejít, ale nebývá to příliš časté.

3.1. HISTORIE

Jazyk Lua vznikl v roce 1993 za účelem zjednodušení práce skupině brazilských inženýrů z naftové společnosti PETROBRAS. Název pochází z portugalského slova *lua*, což v překladu znamená *měsíc*. V prvotní fázi šlo pouze o jednoduchý jazyk, který neobsahoval konstrukce jako podmínky, cykly atd. Postupem času se rozšířil do více oddělení společnosti PETROBRAS, a tím vznikla také poptávka po další funkcionalitě jazyka. (1)

V roce 1994 došlo k vydání Lua verze 1.1, ta se ovšem dala bezplatně využít jen pro akademické účely. Díky tomuto omezení se však Lua příliš neuchytil. Další verze 2.1 byla proto vydána jako free software s vlastní licencí. Od verze Lua 5.0 přišel v šíření jazyka velký zlom – byl vydán pod licencí GPL. Od této doby se začíná prosazovat jak v komerčních, tak i nekomerčních aplikacích.(1)

3.2. SYNTAXE

Jak již bylo řečeno, syntaxe některých konstrukcí je velmi podobná programovacímu jazyku Pascal. Toto je znát hlavně při konstruování struktur podmínek, cyklů aj. I když implicitně nepracuje s objekty, lze vytvářet třídy za pomocí meta-mechanismu. Pro volání metod nebo proměnných těchto tříd se používá tečková notace. Při deklaraci proměnných není třeba definovat datový typ. Používá se zde klíčové slovo *local* stejně jako například *var* v JavaScriptu. Jazyk Lua podporuje také tzv. vícenásobné přiřazení hodnot. Funkce tak může vracet např. tři hodnoty, které se poté přiřadí do tří různých proměnných.(1)

4. JAZYK UML

Před zahájením implementační části bylo potřeba si nejprve ujasnit, jak by se hra měla v určitých situacích chovat, a najít stavy, ve kterých se může hra nacházet. K této analýze jsme použili nástroj, který je již standardem, jazyk UML – Unified Modeling Language.

UML vznikl v roce 1995 sjednocením dvou metod – OMT (Rumbaugh) a OOAD (Booch). Před sjednocením existovalo množství metod, které si byly docela blízko, avšak v některých věcech se rozcházely. K odstranění situací, kdy jeden symbol znamenal pro více lidí něco jiného, začal vznikat jeden společný jazyk.(2)

Jedná se o soubor diagramů, které nám umožňují modelovat navrhovaný systém z různých pohledů:

- *Diagram případů užití*
- *Diagram tříd*
- *Diagram objektů*
- *Diagram komponent*
- *Sekvenční diagram*
- *Stavový diagram*
- *Diagram aktivit*

5. SPRÁVA VERZÍ APLIKACE

U projektů, na kterých pracuje větší množství lidí najednou, může nastat situace, kdy se dva a více lidí bude pokoušet upravit určitou část kódu najednou. K zamezení vytváření těchto konfliktů je potřeba zřídit systém, který by výstupy jednotlivých pracovníků slučoval dohromady. Z mnoha již hotových řešení jsme nakonec vybrali systém Apache SVN neboli Subversion.

Subversion vznikl v roce 2000 jako produkt společnosti CollabNet proto, aby odstranil některé nevýhody staršího systému CVS. Rokem 2009 se stal součástí inkubátoru společnosti Apache Software Foundation. V roce 2010 se místo zavedeného názvu Subversion začal používat název nový – Apache Subversion. (3)

SVN patří mezi centralizované verzovací systémy. To znamená, že veškerá data se ukládají na jeden server, který řídí správu verzí. Klienti tak musí své lokální kopie tohoto serveru aktualizovat. Po provedení změn je opět musí uživatel nahrát na server, aby je měli k dispozici také další uživatelé. Oproti centralizovaným verzovacím systémům pak existují také distribuované systémy. Zde již není centrální server nutný. Veškerá data jsou uložena na klientských počítačích. Mezi nejznámější představitele těchto systémů patří Git, Bazaar a Mercurial. Při nahrávání změn na SVN bývá zvykem, aby uživatel přidal textový popis změn, které provedl. Díky SVN se totiž můžeme vrátit např. k verzi hry, která byla aktuální před půl rokem. Poznámky nám tak umožní se lépe ve změnách orientovat.

S každou provedenou změnou se navýší tzv. číslo revize. Na server se poté neuloží celý nový soubor, ale pouze to, k jakým změnám od poslední revize došlo.

6. HERNÍ ŽÁNRY

Tak jako většina odvětví je i herní svět rozdělen do několika skupin. U her se místo odvětví používá slovní spojení „herní žánr“. Většina uživatelů tak vyhledává hry především podle svého oblíbeného žánru. Proto jsme se před vývojem zaměřili na to, jakým směrem se budeme ubírat – jakého žánru výsledná hra bude. Existuje nepřeberné množství žánrů, a to také díky tomu, že se mezi sebou navzájem kříží. V následujícím textu uvedu některé základní žánry a stručně je popíši.

Adventura – uživatel je zde často v roli jedné nebo více hlavních postav. Pohybuje se ve světě, ve kterém může využít vlastnosti některých předmětů nebo schopností ostatních obyvatel tohoto světa. Aby mohl postupovat dále, je potřeba plnit logické, dovednostní aj. úkoly. Tyto hry tedy mají přesně daný scénář a při opakovaném hraní na hráče nečeká nic nového.(4)

Akční hra – tímto žánrem jsou označovány hry, ve kterých se hráč musí pokusit o zneškodnění nepřátel. Aby to ale nebylo tak jednoduché, jsou nepřátelé vybaveni umělou inteligencí a snaží se mu v tom zabránit. Tyto hry jsou často opatřeny modulem pro umožnění víceuživatelského režimu. Hráč tak má možnost bojovat proti opravdovým protihráčům namísto umělé inteligence.(4)

Arkáda – hry tohoto žánru většinou od hráče vyžadují určitou míru postřehu. Jsou často rozděleny na několik kol, jejichž délka může být časově omezena. Můžeme se setkat s hrami, které si např. upraví některé fyzikální zákony, aby do hry vnesly určitou dávku chaosu.

Strategie – u strategických her je kladen důraz – jak již název napovídá – na strategické dovednosti hráče. Získává k dispozici určitou skupinu objektů (lidí, strojů, továren aj.), kterým musí přidělovat úkoly tak, aby zničil protivníka. Tyto objekty může během hry vylepšovat, musí ale mít dostatek zdrojů (peníze, potrava, dřevo aj.).(4)

Simulátor – u předchozích případů nebyl příliš velký důraz kladen na to, aby hra kopírovala chování v reálném světě. Kvůli tomu je zde herní žánr „simulátor“. Tyto hry se snaží co nejvíce přiblížit reálnému světu. Ať už se jedná o fyzikální zákony nebo co možná nejvěrohodnější grafické ztvárnění. Hráč se tak může ocitnout např. v pilotní kabině a pilotovat dopravní letadlo nebo si zahrát fotbal se svým oblíbeným klubem.(4)

7. PROJEKT RE!SOURCES

Cílem mé diplomové práce bylo vytvořit herní aplikaci spustitelnou na mobilních zařízeních, která bude využívat prvky 3D, a především pak práce s těmito 3D objekty.

Před samotným začátkem vývoje bylo nutné vymyslet celý koncept hry. Nejprve bylo třeba rozhodnout, jakého žánru chystaná hra bude. Mnoho současných titulů nelze zařadit pouze do jednoho konkrétního žánru, ale naopak se žánry mezi sebou navzájem prolínají. Naše hra se ubírá stejným směrem. Proto se nedá říci, že se jedná čistě o arkádu či strategii, ale o jakýsi střed mezi oběma žánry. Hru jsme tedy koncipovali jako multi-level arkádu s prvky real-timové strategie.

7.1. ZÁKLADNÍ KONCEPT

Primárním cílem hry je vypěstovat předem stanovený počet plodů na svém stromě. Snažíme se zde simulovat růst stromu v reálném světě, kde o úspěšném pěstování rozhoduje mnoho faktorů. Hráč může budovat strom pomocí dvou typů objektů – *větve a kořeny***.**

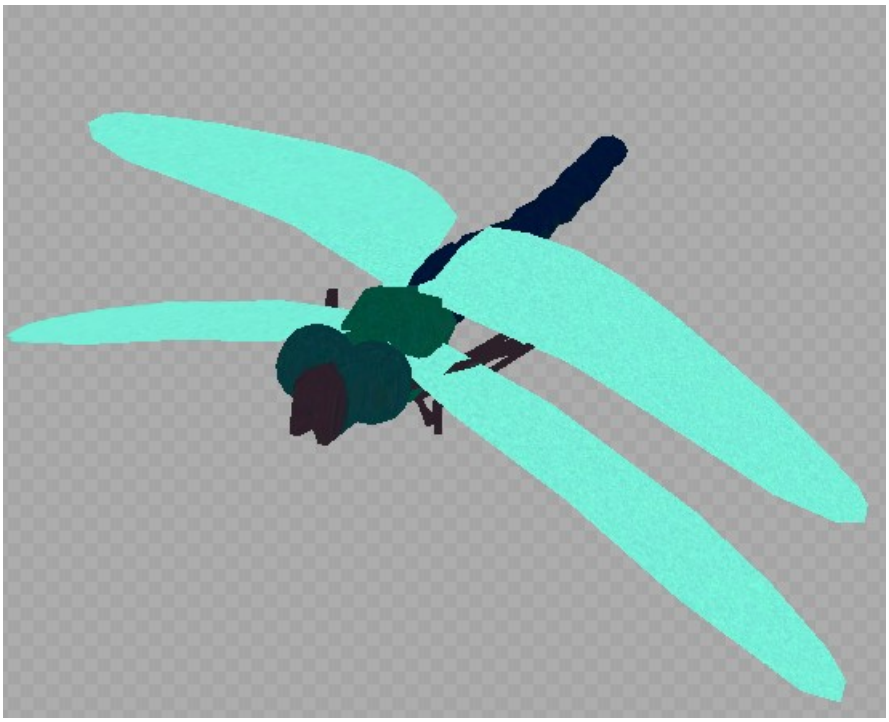
Větve – část stromu, která se umísťuje nad zemský povrch. Každá větev má tu vlastnost, že z okolního světa získává vzduch. Větev, která má na sobě list navíc, získává energii ze slunce. Kromě listů může větev produkovat také květy, které se za nějaký čas přemění na sbírané plody.

Kořeny – oproti větvím jsou kořeny umístěny pod zemským povrchem. Další rozdíl je také v tom, že kořeny nic neprodukují. Pouze získávají energii z půdy, a to buď ve formě vody, nebo živin.

Aby bylo možné stavět opravdu rozmanité stromy, dáváme uživateli na výběr mezi třemi tvary jak kořenů, tak větví, které na sebe navzájem navazují.

To, co strom získává z okolního světa – již zmiňovaný vzduch, světlo, voda a živiny – budeme dále nazývat *zdroji*. Sbíráním těchto zdrojů se navyšuje hodnota celkové energie stromu. Kromě těchto čtyř zdrojů jsme se rozhodli pro implementaci také dvou speciálních, které se zobrazují v celém stromě – jeden, který energii stromu značně navýší, a jeden, který ji naopak snižuje. Celková energie stromu je podle velikosti stromu postupně spotřebovávána. Uživatel je tak nucen neustále doplňovat zásoby energie.

Hlavní postavou hry je živočich (viz obr. 2), který žije uvnitř stromu a pomáhá mu při získávání energie a růstu. Hráč se tímto živočichem pohybuje po celém stromě a sbírá zdroje, které strom získává z okolního světa.



obr. 2 – 3D model vážky

K tomu, aby byl uživatel schopen strom rozšiřovat, je potřeba, aby měl strom určitou energii, kterou získává ze zdrojů. Kromě podmínky minimální energie stromu je uživatel také omezen okolními stromy. Přes větve těchto stromů je zakázáno uživatelův strom rozšiřovat. Pod zemským povrchem nám kromě kořenů okolních stromů brání v rozšiřování překážky ve formě kamenů.

Hra má celkem 10 kol, jejichž projitím uživatel zvítězí. Každé kolo má na začátku stanoven počet plodů, které musí strom vyprodukovat. Jakmile je tato hranice překonána, je uživateli umožněn postup do dalšího kola. Pokud se energie stromu dostane pod hranici, která je specifikována u každého kola, je kolo ukončeno a hráč si je bude muset zopakovat.

7.2. PROSTŘEDÍ HRY

Během celé hry se hráč pohybuje živočichem uvnitř rozrostlého stromu. Větve a kořeny jsou zde znatelně odlišeny – na větvích jsou použity světlejší materiály, kořeny jsou oproti tomu dosti tmavé.

Uživateli jsou k dispozici dva pohledy hry, mezi kterými může libovolně přepínat. V obou pohledech je v horní části obrazovky umístěn ukazatel aktuální energie stromu. Dále je pak tlačítko pro zobrazení hlavního menu a ukazatel počtu plodů, které musí uživatelův strom vyprodukovat, aby úspěšně dokončil kolo.

7.2.1. POHLED MAPY

Tento pohled slouží k tomu, aby měl hráč přehled o tom, jak vypadá jeho strom. Může tak sledovat, kde má vykvetlý květ, vodní zdroj atd. Vidí zde náhled celé herní plochy. Kromě stromu uživatele jsou v tomto pohledu zobrazeny také okolní stromy, zdroje a překážky v půdě. Aby se lépe orientoval v dalším pohledu, ve kterém se může pohybovat skrze celý strom, vidí svou pozici označenou červeným křížkem.

obr. 3 – pohled na mapu

Čas strávený na této obrazovce není určen jen pro statické prohlížení herní plochy, ale také pro správu uživatelského stromu. Uživatel zde má možnost rozšiřovat, ale také upravovat svůj strom. Úpravou se myslí vytváření listů anebo květů, které se umisťují na konce každé ze tří typu větví. Po stisknutí prvního tlačítka, které je umístěno ve spodní části obrazovky, se objeví menu, ve kterém si uživatel smí vybrat buď jeden ze tří tvarů větví/kořenů, nebo umístit na jednu z větví stromu list či květ – má-li na to ovšem dostatek energie a volného prostoru.

Čím více se uživatelův strom rozrůstá, tím je pro hráče obtížnější přejíždění mezi vzdálenějšími větvemi. Pro usnadnění této situace je zde implementován systém navigace. Po stisknutí tlačítka s vlajkou stačí tuto vlajku přetáhnout na větev anebo kořen, do kterého se chce uživatel přesunout. Koncový bod navigace je následně označen vlajkou umístěnou nad ním.

Poslední tlačítko slouží pro přepnutí do druhého herního pohledu, a to do pohledu zevnitř stromu.

7.2.2. POHLED ZEVNITŘ STROMU

Většinu herního času se bude uživatel pohybovat v tomto pohledu. Hlavním úkolem zde je sbírání zdrojů, které strom získává z okolního světa. Spletitou strukturou stromu se uživatel pohybuje živočichem, kterého snímá kamera umístěná za ním. Uživateli jsou k dispozici dva joypady a jedno tlačítko. Levý joypad slouží k natáčení živočicha okolo obvodu větve. Živočich se natočí přesně na tu pozici, kde umístíme prst. Pravý joypad je ve formě vertikálního posuvníku, kterým si uživatel nastaví rychlost, kterou živočich poletí. Jediné tlačítko, které je zde k dispozici, je umístěno pod pravým joypadem a slouží pro otočení živočicha o 180 stupňů nazpět.

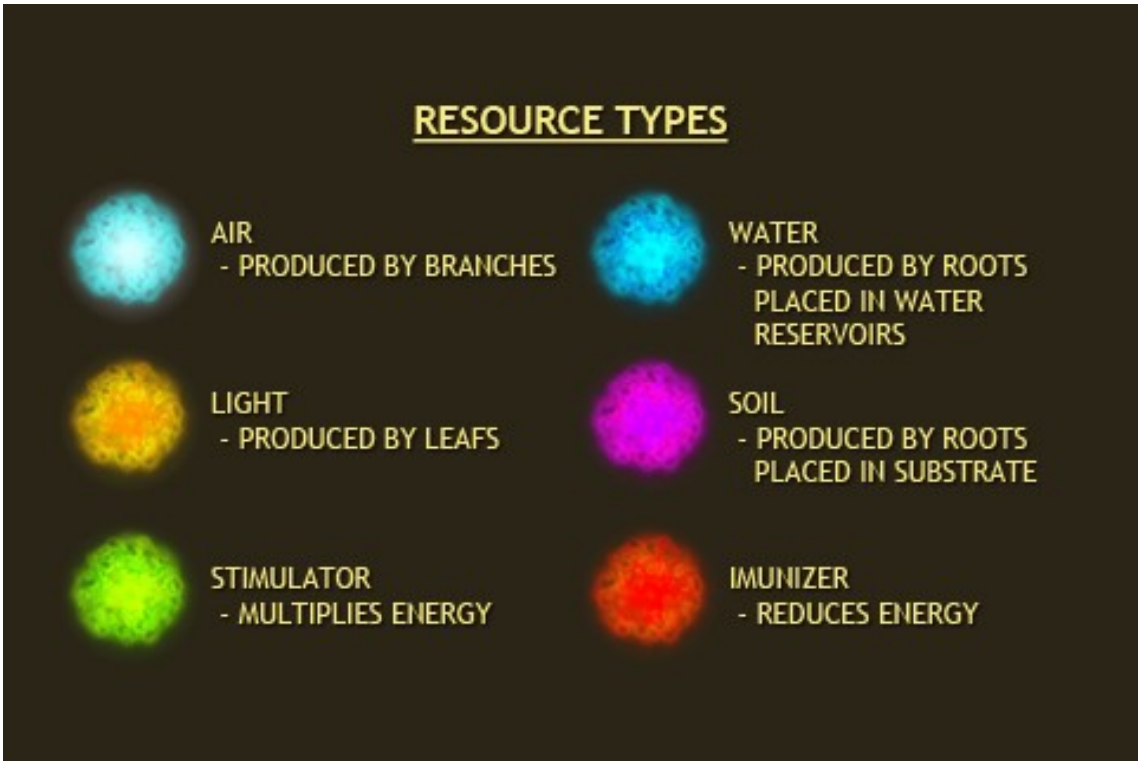
Během přesouvání ve stromu může uživatel narazit na objekty kulatého tvaru. Ty představují zdroje z okolního světa. Uživatel se musí pokusit tyto zdroje sbírat.

Pokud uživatel v pohledu mapy použil systém navigace umístěním vlajky na cílovou větev nebo kořen, budou se mu v místech, kde se strom rozvětvuje, zobrazovat šipky, které mu ukáží cestu k cíli. Jakmile se k němu dostane, zobrazí se v něm pulsující vlaječka.

obr. 4 – pohled zevnitř stromu

7.2.3. ZDROJE

Během své cesty uvnitř stromu uživatel narazí na celkem šest typů objektů. Čtyři z nich symbolizují zdroje získané z okolního světa. Další dva jsou náhodně generované a mohou se zobrazit ve všech částech stromu. Všechny zdroje, které je možné uvnitř stromu sbírat, jsou zobrazeny na obrázku obr. 5.



obr. 5 – náhled zdrojů

7.2.4. POHYB UVNITŘ STROMU

Struktura stromu může být často velmi složitá a hráč se navíc musí pohybovat po docela striktně dané dráze. Proto bylo potřeba hráči co možno nejvíce zjednodušit ovládání hry. S živočichem smí hráč provádět pouze dva základní pohyby:

- *let dopředu*
- *otáčení kolem středové osy větve/kořene*

Velmi často bude ovšem potřeba projet některou z křižovatek – místo, kde dochází k rozvětvení. K tomu, aby odbočil na levou nebo pravou stranu, musí být živočich na tuto stranu dostatečně natočen. Proces zatočení se poté provede automaticky. Pokud by z natočení nebylo jasné, na kterou stranu chce uživatel zatočit, živočich se před křižovatkou zastaví a čeká se, až si uživatel vybere cestu, kterou se vydá.

Jak jsem již zmiňoval, může živočich létat pouze dopředu. Co ale v případě, kdyby se chtěl uživatel vrátit do předchozí větve? K tomu slouží tlačítko pro otočení o 180 stupňů. Po jeho stisknutí se živočich automaticky otočí o 180 stupňů a pokračuje v letu v opačném směru.

7.3. PRŮBĚH HRY

V následujícím textu se pokusím shrnout to, o čem jsem psal doposud. Tak, aby si čtenář uvědomil, že ačkoli se hra zdá na první pohled nesrozumitelná, je ve skutečnosti složena jen z pár snadno zvládnutelných kroků.

1. *Před začátkem samotného hraní hry, by si měl uživatel pečlivě pročíst instrukce. Ty se zobrazí po kliknutí na tlačítko "Instructions" v hlavním menu.*
2. *Po prvním spuštění hry se uživateli zobrazí okno, ve kterém si vybere kolo, které chce hrát – nejprve bude aktivní pouze kolo číslo 1.*
3. *Po načtení vybraného kola se objeví dialogové okno, které hráče upozorní na počet plodů, které je třeba pro úspěšné dokončení kola vyprodukovat.*
4. *Jakmile uživatel dialogové okno potvrdí, zobrazí se mu základní pohled mapy.*
5. *Nyní má uživatel dvě možnosti:*
    a. *Rozmístit nové větve, listy nebo plody.*
    b. *Přepnout se do pohledu zevnitř stromu a sbírat energii, díky které může stavět.*
6. *Po rozmístění stanoveného počtu plodů kolo končí a uživateli se objeví okno pro výběr dalšího kola.*

8. PŘÍPRAVA SERVERU

Před zahájením samotné práce na návrhu a implementaci hry jsme museli připravit prostředí, ve kterém budeme naše výstupy ukládat. Jako server bylo použito kompletní řešení od společnosti Fujitsu Siemens se dvěma 500 GB disky zapojenýma v RAID 1 poli. V případě poruchy jednoho disku tak nehrozí ztráta dat, protože je druhý disk zrcadlovým obrazem prvního.

S ohledem na pořizovací náklady a také na jednoduchost konfigurace jsme jako operační systém zvolili OS Linux, konkrétně v distribuci Fedora 14, která byla v době instalace serveru aktuální. Žádná ze služeb nevyžadovala zvláštní zabezpečení, protože přístup na server je povolen pouze přes VPN spojení zabezpečeného pomocí IPSec. O toto zabezpečení se stará router, na který je server připojen.

Na serveru bylo potřeba nastavit tyto služby:

- *SSH*[10] *pro vzdálenou správu*
- *Webový server Apache*[11]
- *Apache Subversion*
- *Systém pro správu chyb – BugZilla*[12]
- *Sdílený adresář – Samba server*[13]

[10] http://www.openssh.org/
[11] http://www.apache.org/
[12] http://www.bugzilla.org/
[13] http://www.samba.org/

9. NÁVRH A IMPLEMENTACE

Jak už bylo řečeno, budu se dále zabývat samotným vývojem aplikace. Měl na starosti zejména tu část hry, ve které se bude koncový uživatel nejvíce pohybovat – 3D prostředí. Pracoval jsem tedy na manipulacích s různými 3D objekty, jejich integraci a interakcích mezi nimi. Další text bude vždy rozdělen na dvě až tři části. V první popíši problém, který bylo třeba vyřešit. Druhá část se bude zabývat návrhem a poslední, třetí část, bude představovat konečnou implementaci. U některých jednodušších problémů může být ihned z návrhu jasné, jakým způsobem by byl implementován. U těchto problémů proto implementační část vynechám.

Přestože Lua sám o sobě nepodporuje objektově orientované programování, je možné na strukturu naší aplikace nahlížet jako na skupinu několika objektů, které spolu komunikují. V Shiva 3D engine jsou totiž použity tzv. AI modely, které představují klasické třídy, jak je známe z OOP. Kromě toho, že musí existovat minimálně jeden centrální model, který bude obsluhovat hru jako celek, musí mít AI model každý 3D objekt, se kterým je potřeba nějakým způsobem komunikovat.

Každý AI model může obsahovat atributy různých datových typů, metody a také handlery, jejichž kód je prováděn, když dojde k nějaké události (např. stisknutí tlačítka na klávesnici apod.). To je ale z podobností s OOP jazyky vše. Nejsou zde možnosti k využívání jak dědičnosti, tak polymorfismu, ani zapouzdření. Ke všem atributům a handlerům lze přistupovat veřejně. K metodám oproti tomu přistupovat přímo z jiných tříd nelze. V praxi se tedy nejčastěji využívá způsob, kdy je ze vzdálené třídy zavolán některý z handlerů. Uživatel má možnost vytvoření vlastních handlerů, ve kterých se provede buď spuštění některé metody, nebo čistě nastavení některé z proměnných.

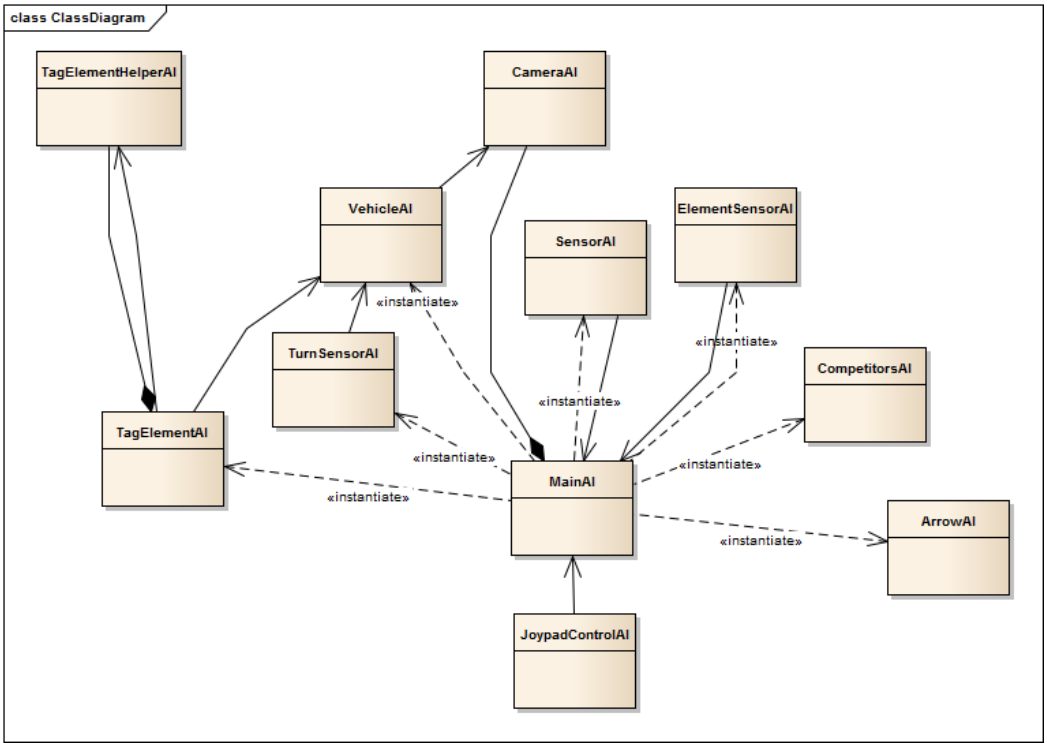
K nastavení atributů nebo zavolání metod uvnitř třídy se používá klasické klíčové slovo *this*, za kterým se napíše tečka, a poté vlastní název atributu nebo metody. Shiva engine obsahuje kolekci základních tříd, jako jsou *object*, sloužící právě ke komunikaci mezi objekty nebo např. pro transformaci objektů. Mezi další hojně používané třídy patří např. *application*. Tato třída obsahuje metody pro zjištění informací o spuštěné aplikaci. Můžeme zde najít metodu, která zjistí, zda je aplikace natočená na výšku nebo na šířku, rozlišení okna, ve kterém je aplikace spuštěna aj. Dále jsou zde vytvořeny třídy pro práci s matematickými funkcemi nebo s řetězci. Z hlediska přístupu k atributům a metodám si je můžeme představit jako statické třídy.

Pro nastavení atributu *bTestAttr* na *false* u objektu *testObject*, který má k sobě přiřazen AI model TestAI se provede následující metoda:

```
object.setAIVariable(testObject, „TestAI“, „bTestAttr“, false)
```

9.1. TŘÍDNÍ DIAGRAM APLIKACE

Na obrázku obr. 6 je k vidění odlehčená verze třídního diagramu celé aplikace. Je z něj patrné, že hlavní třída MainAI vytváří instance všech ostatních tříd, kromě JoypadControlAI. Tu inicializuje Shiva 3D engine při spuštění hry, stejně jako samotnou MainAI.



obr. 6 – třídní diagram aplikace

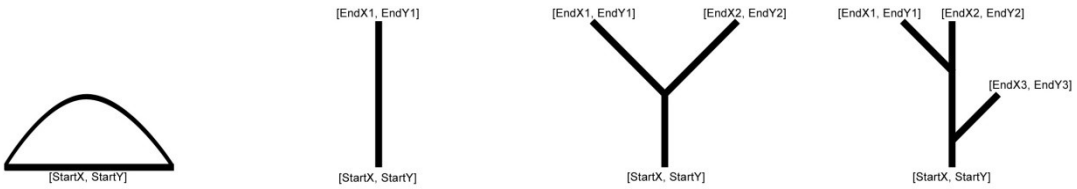
9.2. 3D MODEL STROMU

Jedním z prvních úkolů bylo vytvoření modelu stromu podle struktury, která je při spuštění hry načtena z XML souboru a poté uložena do paměti. Formát XML souboru je následující:

```
<Level>
    <Variables>
      <IDLevel>1</IDLevel>
      <FruitsToWin>3</FruitsToWin>
    </Variables>
    <Trees>
      <Tree IDTree="1" … >
        <Elements>
            <Element IDElement="1" TypeElement="2" MasterType="1"
            Angle="0" StartX="75. 0" StartY="50.0" EndX1="71.3"
            EndY1="57.6" EndX2="78.6" EndY2="57.6" EndX3="75.0"
            EndY3="54.0" EndX4="0.0" EndY4="0.0" EndX5="0.0"
            EndY5="0.0" Surface="2TunnelMatDamaged" />
            <Element IDElement="7" TypeElement="2" MasterType="2" .../>
```

```
        </Elements>
        <Strokes>...</Strokes>
        <SubElements>...</SubElements>
      </Tree>
    </Trees>
    <Areas>...</Areas>
    <MapElements>...</MapElements>
</Level>
```

Každá úroveň tedy obsahuje strukturu, která je podobná výše zmíněnému kódu. Kořenový element <*Level*> v sobě uchovává informace o jedné konkrétní úrovni, dále pak seznam stromů a jejich větví. Pro vykreslení 3D modelu je potřeba pouze strom s ID 1. Další dva stromy jsou konkurenční a jejich růst uživatel neřídí. Každá větev, v našem případě je definována elementem <*Element*>, je identifikovatelná pomocí jedinečného atributu *IDElement*. Jedinečný znamená, že žádná jiná větev jednoho stromu nemůže mít atribut *IDElement* stejné hodnoty. Dalším důležitým atributem je *TypeElement*, ten nabývá číselných hodnot 0-3 a označuje tvar větve, která má být vykreslena. Typ 0 je speciálním tvarem, který se umisťuje na konce listových elementů celé stromové struktury. Listové elementy v tomto případě představují elementy, které nemají žádné potomky. Elementy s typem 0 tedy slouží pouze k zakrytí děr na koncích stromu – jedná se o jakousi zátku. Atribut *MasterType* slouží k identifikaci toho, zda se jedná o stromovou část nad zemí – tedy větve – nebo pod zemí – tedy kořeny. Opět se jedná o číselný atribut, který nyní nabývá hodnot 0-1. Atribut *Angle* určuje natočení elementu oproti ose rodičovského elementu. Další atributy slouží k určení souřadnic konců jednotlivých elementů. Každý element má počáteční souřadnice *StartX* a *StartY*. O existenci dalších atributů s názvem *EndX** a *EndY** rozhoduje tvar elementu. Přípustné hodnoty u jednotlivých tvarů je možné nalézt na obrázku obr. 7.

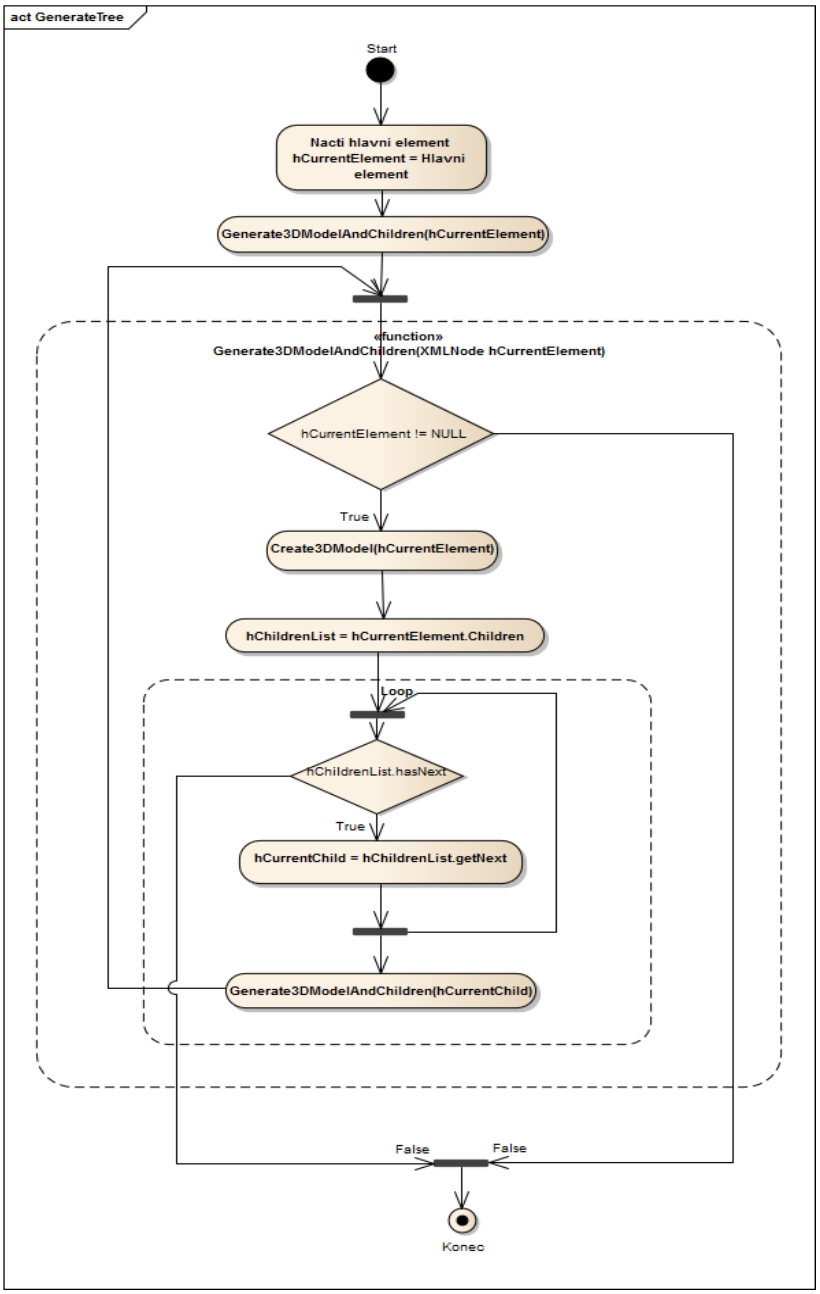


obr. 7 – náhled všech typů větví

9.2.1. NÁVRH

Tuto část bych rozdělil do dvou dílčích celků. Jeden se bude zabývat procházením struktury stromu a druhý samotným vytvořením jedné konkrétní větve.

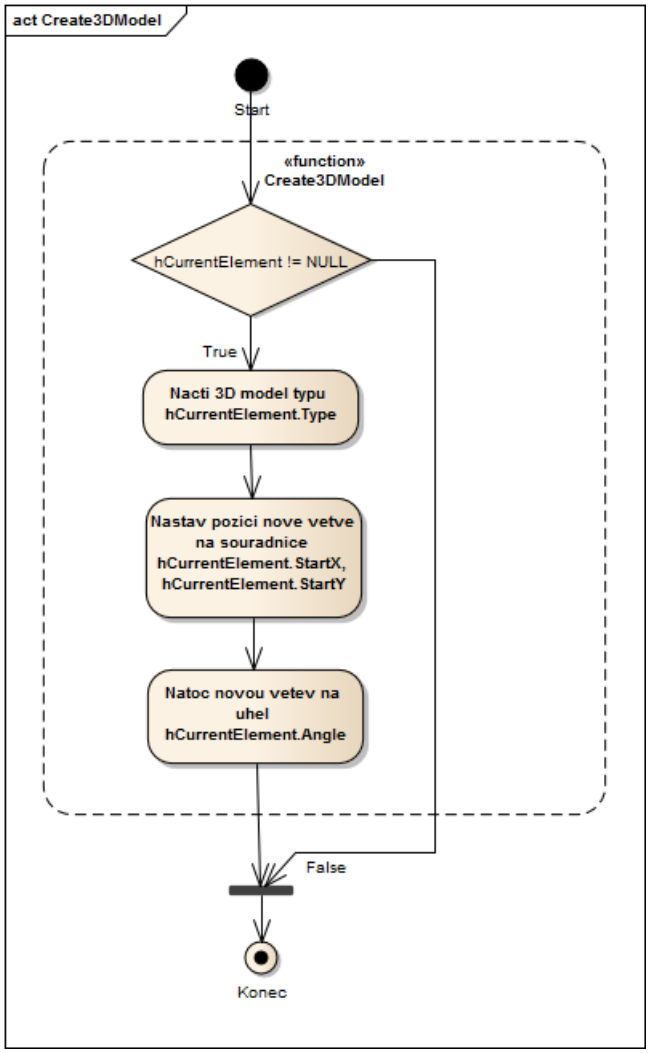
V prvním případě je potřeba zajistit, aby se rekurzivně volala metoda, které vykreslí jednu větev a zároveň vyvolá vytvoření potomků této větve. Návrh tohoto postupu je zobrazen v diagramu aktivit na obrázku obr. 8.



obr. 8 – aktivitní diagram vytváření 3D modelu stromu

Tento diagram řeší vytvoření celé struktury stromu. Postupem času jsme ale zjistili, že při vytváření stromu, který obsahuje příliš mnoho větví, měl fakt zobrazení takto velkého počtu objektů na scéně negativní vliv na výkon celé aplikace. Vytváření 3D stromu bylo potřeba optimalizovat. Konečná aplikace tedy mechanismus generování stromu vylepšila. Negeneruje se najednou celá stromová struktura, ale pouze větev, ve které se nachází vážka včetně všech sousedních větví. Při přejíždění mezi větvemi se tady dynamicky generují nové větve a ty staré zanikají. V návrhu to znamená změnu v tom, že se vykreslí potomci větve, ve které se vážka nachází, samotná větev a poté předek této větve, pokud existuje.

Projití celé struktury máme tedy za sebou a je načase se podívat na vytvoření modelu jedné konkrétní větve, její umístění a natočení v prostoru. Následující diagram se zabývá pouze vytvořením modelu jedné konkrétní větve a jejího umístění. O dalších věcech jako je rozmístění senzorů, modelů zdrojů apod. se zmíním později.



obr. 9 – aktivitní diagram vytváření 3D modelu větve

9.2.2. IMPLEMENTACE

V této části se budu zabývat implementací výše zmíněného návrhu. Místo toho, abych zde zobrazoval zdrojové kódy psané v jazyce Lua, rozhodl jsem se, že všechny ukázky zdrojového kódu přepíši do pseudokódu, aby s jeho pochopením neměl problémy čtenář, který jazyk Lua příliš neovládá.

K vytvoření dokonalého modelu hráčova stromu bylo nutné získat 3D modely jednotlivých větví. K snadnému rozmísťování pak bylo potřeba, aby každý model měl nastaven pivot na polohu, která odpovídá pozici *[StartX*, *StartY]* u jednotlivých typů větví. Díky tomu bude každý potomek určité větve vždy přesně doléhat na jeden z konců svého předka. V následujícím pseudokódu je znázorněno vytváření jednotlivých 3D modelů větví.

```
If current element changed then

      For all last element child and
               parent except current element
      do
          Remove element 3D model
      Endfor

      For all current element child except last element do
            Create 3D model tmpModel with type currentItem.Type
            Move tmpModel to[currentItem.StartX, currentItem.StartY]
            Rotate tmpModel to currentItem.Angle
      Endfor

Endif
```

Při přejetí do nové větve se odstraní všechny sousední větve naposledy navštívené větve, kromě té, do které uživatel přejel. Poté se provede vytvoření 3D modelů větví, které soused s větví, ve které se uživatel právě nachází. V tomto případě se nezavolá metoda pro vytvoření poslední navštívené větve, protože ta nebyla v předchozím kroku smazána.

9.3. ROZMÍSTĚNÍ SENZORŮ

Senzor je v Shiva 3D enginu speciální objekt, který slouží k tomu, aby při kolizi s jiným senzorem vyvolal událost. Senzorem mohou být dva typy objektů: *kvádr a koule*. Lze s nimi provádět operace běžné pro ostatní 3D modely jako je přesun, rotace atd. Oproti ostatním objektům však senzory nejsou na scéně vidět (kromě testování aplikace v prostředí Shiva 3D editoru). Senzor je také možné připojit k jinému 3D objektu. Tuto možnost používáme například u zdrojů, které na sobě mají umístěný senzor proto, aby bylo možné zjistit, že s nimi vážka kolidovala, a zajistit jejich sběr. Rozmístění zdrojů však proberu do detailů později. Nyní se zaměřím spíše na senzory, které nejsou přilinkovány k jinému objektu.

Existují hned tři typy handlerů, které události vyvolané při kolizi senzorů odchytávají:

- *onSensorCollisionBegin*
- *onSensorCollision*
- *onSensorCollisionEnd*

Z názvů jednotlivých handlerů lze snadno odhadnout, na jakou událost reagují, přesto si je krátce popíšeme.

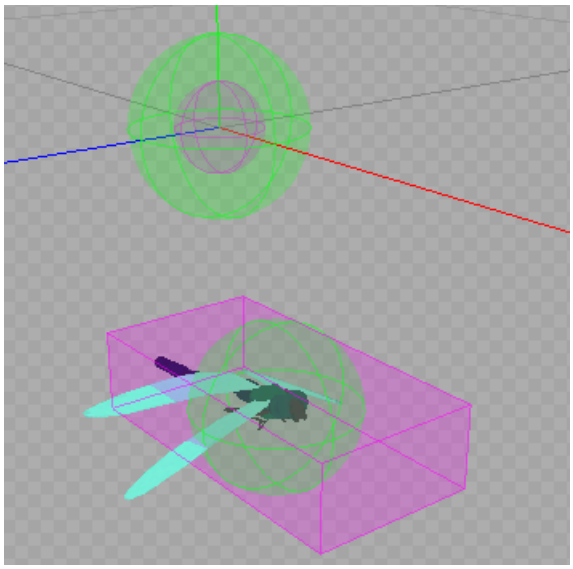
onSensorCollisionBegin – tento handler je spouštěn při prvotním kontaktu dvou senzorů. To znamená, že se při kolizi zavolá pouze jednou.

onSensorCollision – na rozdíl od prvního handleru, je tento při kolizi zavolán tolikrát, kolikrát proběhne obnovení obrazovky, během toho, kdy jsou dva senzory v kontaktu. Kdyby tedy byly dva senzory v kontaktu po dobu pěti framů, tak by se tento handler zavolal pětkrát.

onSensorCollisionEnd – handler *onSensorCollisionEnd* se chová prakticky stejně jako handler *onSensorCollisionBegin*. Jediný a hlavní rozdíl je v tom, že namísto prvotního kontaktu s dalším senzorem je tento handler zavolán, když dojde k ukončení kontaktu mezi dvěma senzory.

V projektu Re!Sources jsme se omezili hlavně na používání *onSensorCollisionBegin* a *onSensorCollisionEnd*, protože pro naše potřeby byly dostačující a navíc volání handleru *onSensorCollision* je prováděno každý frame, takže by docházelo ke snížení výkonu aplikace.

Aplikace používá tři typy senzorů, které se dynamicky rozmisťují po celém stromě. Jeden, který slouží k označení začátku a konce jednotlivých větví, další k určení místa, kde se nachází křižovatka, a vážce tak bude umožněno zatáčení, a posledním typem je senzor, který upozorňuje na konec stromu. Vážka se při kolizi s tímto senzorem musí zastavit, aby nevyletěla ven. Jak jsem zmínil výše, aby senzor vyvolal některou z událostí, je potřeba, aby kolidoval s jiným senzorem. Kvůli této skutečnosti jsme umístili dva senzory také na model vážky. Protože se vážka otáčí kolem středové osy jednotlivých větví, je jeden senzor umístěn tak, aby se pohyboval po této středové ose, a má tvar koule. Druhý senzor je kvádr a obklopuje viditelný model vážky. Slouží tak ke kolizím s různými zdroji. Pro lepší představu rozmístění senzorů na vážce se můžete podívat na obrázek obr. 10.



obr. 10 – model vážky se senzory

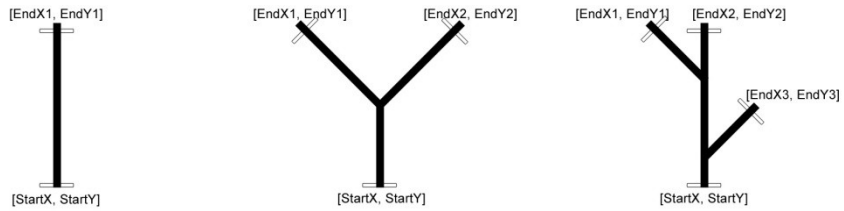
O přesné funkčnosti jednotlivých senzorů se zmíním později. Pro tuto chvíli nám budou stačit tyto základní informace a budeme se zajímat spíše o rozmístění senzorů na správná místa.

9.3.1. NÁVRH

Jak bylo řečeno, aplikace používá tři typy senzorů. Tyto senzory jsou vytvářeny zároveň s modely jednotlivých větví a podle typu větve se také určí počet a pozice senzorů. Nyní ujasníme, kde je potřeba jednotlivé senzory umístit.

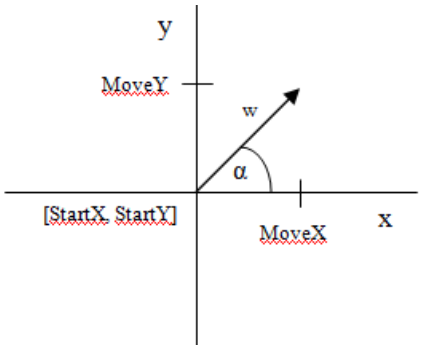
Začneme pro implementaci nejjednodušším senzorem, který se umístí na konce stromu. Jako tvar tohoto senzoru jsme vybrali kvádr, který bude mít velikost takovou, aby zaručovala kolizi s jedním ze senzorů na modelu vážky. Senzor je umístěn na souřadnici *[StartX, StartY]* všech tzv. zátkových elementů – elementů, které představují v celé stromové struktuře listy (nemají žádné potomky). Protože mají tyto senzory tvar kvádru, bylo nutné je natočit na úhel totožný s úhlem zátkového elementu.

Nyní popíši senzor, který slouží k identifikaci větve, ve které se právě nachází vážka. Tento senzor je tvarově totožný s předchozím případem. Senzor se bude vyskytovat v každé větvi vícekrát, kromě zátkového elementu, ve kterém se tento senzor nevygeneruje vůbec. Bude záležet na tom, kolik bude mít větev zakončení. První bude umístěn na souřadnici *[StartX, StartY]* a další budou posunuty od ostatních koncových souřadnic větve o jednu délkovou jednotku směrem ke středu větve. Na obrázku obr. 11 je možné nahlédnout na umístění senzorů u jednotlivých typů větví.



obr. 11 – rozmístění počátečních a koncových senzorů

Pro výpočet souřadnic posunutých senzorů jsem využil vlastnosti Pythagorovy věty v pravoúhlém trojúhelníku. Souřadnici posunutého senzoru vypočteme tak, že budeme brát délku posunutí jako velikost vektoru a ten následně rozložíme pod úhlem větve na složky *X* a *Y*. Výslednou pozici zjistíme tak, že jednotlivé složky přičteme k pozici určující koncový bod větve. Výpočet tohoto posunutí znázorním na obrázku obr. 12 a následující rovnici. Koncové souřadnice jsou v něm znázorněny hodnotami *StartX*, *StartY*.



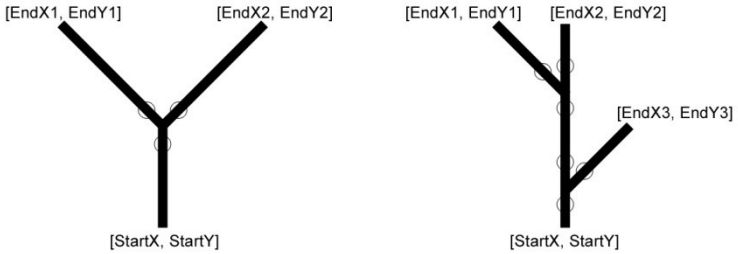
obr. 12 – naznačení posunutí pod úhlem α

$$New[x] = Start[x] + (|w| \cdot sin\,\alpha)$$

$$New[y] = Start[y] + (|w| \cdot cos\,\alpha)$$

O významu tohoto způsobu rozmístění senzorů se opět zmíním později při popisování funkčnosti jednotlivých senzorů.

Posledním typem je senzor, který bude umístěn na křižovatkách uvnitř větví. Nebude se vyskytovat ve všech větvích, ale pouze ve větvích typů 2 a 3, pouze tyto větve totiž křižovatky obsahují. Tvar senzoru bude koule. Každá křižovatka bude obsahovat tři senzory, které budou vzdáleny o jednu délku od středu křižovatky na všechny směry, kterými se může vážka vydat (v našem případě obsahuje každá křižovatka právě tři směry, proto se vytváří také tři senzory). K rozmístění těchto senzorů použiji výpočet směrového vektoru mezi bodem, který určuje střed křižovatky a bodem určeným jedním z koncových bodů větve. Z toho se poté vypočte jednotkový vektor, který se vynásobí délkou, o kterou chceme senzor posunout. V našem případě se jedná o jednu délku. Rozmístění senzorů je znázorněno na obrázku obr. 13.



obr. 13 – rozmístění senzorů na křižovatkách

9.3.2. IMPLEMENTACE

Vytváření senzorů je spojeno s vytvářením jednotlivých větví. Při zavolání funkce, která vygeneruje model větve, se vyhodnotí, o jaký typ větve se jedná, a podle toho se zavolají funkce, které vytvářejí jednotlivé senzory.

```
Load Element information into hElement
if hElement.Type is equal to 0
      create plug element sensor at axis
            hElement.StartX, hElement.StartY with
            hElement.Angle angle rotation
else
      …
```

Nejprve se zkontroluje, zda se jedná o větev typu 0 – větev umístěná na všech koncích stromu. Pokud ano, vytvoří se 3D model větve, umístí se na dané souřadnice a natočí se na úhel *hElementAngle*.

```
if hElement.Type is equal to 0
      …
else
      create first element sensor at axis
      hElement.StartX, hElement.StartY with
            hElement.Angle angle rotation

      set nSecondSensorAxisX to hElement.EndX1 +
      (-1 * sin(hElement.FirstChild.Angle))

      set nSecondSensorAxisY to hElement.EndY1 +
      (-1 * cos(hElement.FirstChild.Angle))

      create second element sensor at axis
            nSecondSensorAxisX, nSecondSensorAxisY
```

Pro všechny ostatní větve (typ 1–3) se na počáteční souřadnici vytvoří první senzor. Na souřadnicích, které jsou o jednu jednotku posunuty od souřadnic *hElement.EndX1* a *hElement.EndY1* (viz návrh), se zase vytvoří druhý senzor.

```
if hElement.Type is equal to 2
      set nThirdSensorAxisX to hElement.EndX2 +
      (-1 * sin(hElement.SecondChild.Angle))
      set nThirdSensorAxisY to hElement.EndY2 +
      (-1 * cos(hElement.SecondChild.Angle))
      create third element sensor at axis
            nThirdSensorAxisX, nThirdSensorAxisY
```

Pokud se jedná o větev typu 2, dojde nejprve k vygenerování posledního ze senzorů, které určují počátek a konec větve. V tomto případě se umístí na souřadnice, které jsou posunuté o jednu jednotku směrem dovnitř větve od souřadnic *hElement.EndX2* a *hElement.EndY2*.

```
      set nDirectionalVectorX to hElement.StartX - hElement.EndX3
      set nDirectionalVectorY to hElement.StartY - hElement.EndY3
      set nDirectionalVectorLength to
            sqrt(nDirectionalVectorX^2 + nDirectionalVectorY^2)

      set nUnitVectorX to
            nDirectionalVectorX / nDirectionalVectorLength

      set nUnitVectorY to
            nDirectionalVectorX / nDirectionalVectorLength

      set nSensorAxisX to hElement.EndX3 +
                              1 *  nUnitVectorX
      set nSensorAxisY to hElement.EndY3 +
                              1 *  nUnitVectorY

      create first turn sensor at axis
            nSensorAxisX, nSensorAxisY

      set nDirectionalVectorX to
            hElement.EndX1 - hElement.EndX3

      set nDirectionalVectorY to
            hElement.EndY1 - hElement.EndY3

      […  count unit vector …]

      set nSensorAxisX to hElement.EndX3 +
                              1 *  nUnitVectorX
      set nSensorAxisY to hElement.EndY3 +
                              1 *  nUnitVectorY

      create second turn sensor at axis
            nSensorAxisX, nSensorAxisY

      set nDirectionalVectorX to hElement.EndX2 - hElement.EndX3
      set nDirectionalVectorY to hElement.EndY2 - hElement.EndY3
      […  count unit vector …]
```

```
            set nSensorAxisX to hElement.EndX3 + 1 *  nUnitVectorX
            set nSensorAxisY to hElement.EndY3 + 1 *  nUnitVectorY
            create third turn sensor at axis nSensorAxisX, nSensorAxisY
```

V právě provedených krocích došlo k výpočtu směrového a následně jednotkového vektoru, díky kterým pak můžeme říci, jakým směrem se posune jeden z křižovatkových senzorů. Toto se provede třikrát, protože větev typu 2 obsahuje právě tři senzory na křižovatce. Pro získání konečných souřadnic se přičtou souřadnice jednotkového vektoru k souřadnici středu křižovatky. Tyto kroky se zopakují postupně pro všechny tři křižovatkové senzory.

U posledního typu větve je vytváření senzorů velmi podobné tomu u větve typu 2. Proto zde již nebudu vypisovat pseudokód. Stejně jako u větve typu 2 se nejprve naleznou souřadnice všech koncových senzorů, které větev obsahuje. Senzory jsou nyní dva a jsou posunuty o jednu jednotku od souřadnic *hElement.EndX2, hElement.EndY2* a *hElement.EndX3, hElement.EndY3*.

Posledním krokem je vytvoření senzorů na křižovatkách. Oproti předchozí větvi je situace složitější pouze v tom, že větev typu 3 obsahuje dvě křižovatky. Celkem je tedy potřeba vytvořit šest senzorů. Samotný výpočet souřadnic senzorů je pak shodný s výpočtem souřadnic u předchozích větví.

9.4. FUNKCE SENZORŮ

O typech senzorů jsem se zmínil již dříve. V této kapitole se zaměřím na to, co se stane, když dojde ke kolizi vážky s jednotlivými senzory. Některé detaily, které ovlivňují pohyb vážky, si nechám na jednu z příštích kapitol a nyní se zmíním pouze o tom, jaký pohyb bude proveden, nikoli o tom, jak bude pohyb proveden.

9.4.1. ANALÝZA

V analýze proberu možnosti, které mohou nastat při kontaktu vážky s některým senzorem. Bylo tedy potřeba ošetřit následující situace:

- *Zastavení vážky na konci stromu*
- *Zjištění větve, ve které se vážka nachází – z důvodu dynamického generování sousedních větví*
- *Zatáčení vážky na křižovatkách*

Z těchto tří možností se budu nejvíce zaobírat právě tou poslední, která je k pochopení principu nejnáročnější.

9.4.1.1. ZASTAVENÍ NA KONCI STROMU

Řešení tohoto problému bylo velmi snadné. V zásadě se jednalo o to, aby se při kolizi se senzorem, který určuje konec stromu, nastavila proměnná, která zastaví pohyb vážky na hodnotu *true*. Pokud se vážka otočí do protisměru, změní se hodnota této proměnné na *false*. Pokaždé, když se tedy vážka, která koliduje s koncovým senzorem, otočí o 180°, provede se negace hodnoty této proměnné.

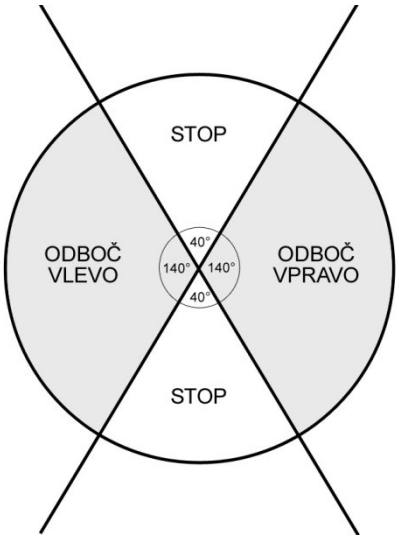
9.4.1.2. URČENÍ AKTUÁLNÍ VĚTVE

Problém určení aktuální větve, ve které se vážka nachází, je řešen pomocí senzorů, které jsou umístěny na startovních a koncových pozicích každé větve. Během projíždění stromem bude vážka kolidovat se senzory v určitém pořadí. Nejprve dojde ke kolizi se senzorem, který patří větví, ve které se vážka nachází. Při kolizi s druhým senzorem dojde ke změně identifikátoru aktuální větve na identifikátor větve, které druhý senzor patří, a k vygenerování větví, které sousedí s aktuální větví, ostatní se ze scény vymažou. Může zde dojít k situaci, že se vážka otočí po projetí prvního senzoru do protisměru a následně bude kolidovat se stejným senzorem. V takovém případě nedojde ke generování nových a smazání starých větví.

9.4.1.3. ZATÁČENÍ NA KŘIŽOVATKÁCH

Část, která se zabývala ovládáním zatáčení vážky na křižovatkách, byla jednou z nejnáročnějších vůbec. Bylo zde třeba ošetřit poměrně velké množství situací, které při průletu stromem mohou nastat. Senzory jsou umístěny ve všech směrech křižovatky. Pokud chce tedy uživatel proletět křižovatkou, musí dojít ke kolizi právě se dvěma senzory. Samotný průlet mezi dvěma senzory je prováděn automaticky, pokud vážka po průletu křižovatkou změní úhel svého natočení. U křižovatek větve typu 3 lze totiž projet obě křižovatky rovně, čili není potřeba korigovat pohyb vážky tak, ať se postupně natáčí na nový úhel.

O tom, na jakou stranu má vážka zatočit, se rozhodne podle toho, pod jakým úhlem bude nakloněna vzhledem ke svislé ose větve. Pokud bude natočena na jakoukoli stranu pod úhly 20°–160°, provede se zatočení. V opačném případě se vážka zastaví a počká se na reakci uživatele, který se může otočit do protisměru a pokračovat v jízdě nebo se natočit s vážkou na stranu, na kterou chce odbočit, a pokračovat tímto směrem. Názorněji to bude zobrazeno na obrázku obr. 14.



obr. 14 – rozdělení natočení vážky

V případě, že je možné proletět křižovatkou rovně (větev typu 3), vozítko se před křižovatkou nikdy nezastaví. Když bude v tomto případě natočeno na úhel, který v předchozím obrázku znázorňuje STOP, vážka bude pokračovat průletem křižovatky rovně.

Každý senzor v sobě bude uchovávat informace o souřadnicích okolních senzorů. Toto je použito k tomu, aby se po průletu křižovatkou nastavila pozice vážky na souřadnice koncového senzoru. To nám zajistí, že bude vážka po průletu křižovatkou opět ve středové ose větve a nehrozí, že by při svém pohybu z této osy sjela. Pokud tedy bude vážka natočena na pravou stranu, použijí se souřadnice pravého senzoru. Kromě těchto informací obsahují také hodnotu úhlu, o který se má vážka na konkrétní stranu natočit.

Bylo potřeba zajistit, aby vážka proletěla křižovatkou plynule. Její pohyb bude tedy opisovat kružnici opsanou trojúhelníkem, který je dán body počátečního senzoru, koncového senzoru a středu křižovatky. Pro výpočet středu této kružnice, který následně bude bodem, kolem kterého se bude vážka otáčet, potřebujeme najít alespoň dva vektory, které spojí všechny tři body trojúhelníku. Vytvoříme tedy vektory, které spojují počáteční senzor se středem křižovatky, a také vektor, který spojuje střed křižovatky s koncovým senzorem. Nejprve získáme směrové vektory mezi jednotlivými body:

Start … souřadnice startovního senzoru

Center … souřadnice středu křižovatky

End … souřadnice koncového senzoru

$\mathbf{SV_1}$ … směrový vektor mezi body Start a Center

$\mathbf{SV_2}$ … směrový vektor mezi body Center a End

Výpočet směrového vektoru mezi body Start a Center:

$$\mathrm{SV}_1[\mathrm{x}] = \mathrm{Center}[\mathrm{x}] - \mathrm{Start}[\mathrm{x}]$$

$$\mathrm{SV}_1[\mathrm{y}] = \mathrm{Center}[\mathrm{y}] - \mathrm{Start}[\mathrm{y}]$$

Výpočet směrového vektoru mezi body Center a End:

$$\mathrm{SV}_2[\mathrm{x}] = \mathrm{End}[\mathrm{x}] - \mathrm{Center}[\mathrm{x}]$$

$$\mathrm{SV}_2[\mathrm{y}] = \mathrm{End}[\mathrm{y}] - \mathrm{Center}[\mathrm{y}]$$

Dalším krokem je nalezení středů těchto vektorů. Vypočteme tedy velikost obou vektorů, vydělíme ji dvěma a výsledek přičteme k souřadnicím bodů *Start*, resp. *End*. Z těchto středů poté povedeme kolmice. Tam, kde se kolmice protnou, vznikne střed kružnice opsané.

$\mathbf{S_1}$ … souřadnice středu strany |StartCenter| trojúhelníku

$\mathbf{S_2}$ … souřadnice středu strany |CenterEnd| trojúhelníku

Velikost vektoru SV_1:

$$|\mathrm{SV}_1| = \sqrt{\mathrm{SV}_1[\mathrm{x}]^2 + \mathrm{SV}_1[\mathrm{y}]^2} \quad (5)$$

Velikost vektoru SV_2:

$$|\mathrm{SV}_2| = \sqrt{\mathrm{SV}_2[\mathrm{x}]^2 + \mathrm{SV}_2[\mathrm{y}]^2} (5)$$

Střed strany |StartCenter|:

$$\mathrm{S}_1[\mathrm{x}] = \mathrm{Start}[\mathrm{x}] + \frac{|\mathrm{SV}_1|}{2}$$

$$\mathrm{S}_1[\mathrm{y}] = \mathrm{Start}[\mathrm{y}] + \frac{|\mathrm{SV}_1|}{2}$$

Střed strany |CenterEnd|:

$$S_2[x] = \text{Center}[x] + \frac{|SV_2|}{2}$$

$$S_2[y] = \text{Center}[y] + \frac{|SV_2|}{2}$$

Kolmice na vektor se provede vytvořením tzv. normálového vektoru. Stačí tedy prohodit souřadnice a u jedné změnit znaménko.

Směrový vektor u: (x, y)

Normálový vektor k vektoru u: $(-y, x)$ (6)

$\mathbf{NV_1}$ … vektor osy první strany trojúhelníku

$\mathbf{NV_2}$ … vektor osy druhé strany trojúhelníku

$$NV_1 = (SV_1[y], -SV_1[x])$$

$$NV_2 = (SV_2[y], -SV_2[x])$$

Posledním krokem je nalezení průsečíků těchto normálových vektorů. V tomto průsečíku se bude nacházet střed kružnice opsané. Nejprve si vyjádříme jednotlivé vektory obecnou rovnicí přímek a poté provedeme výpočet souřadnic středu kružnice pomocí determinantů.

Obecná rovnice přímky:

$ax + by + c = 0$ (6)

$\mathbf{S_1}$ … souřadnice středu první strany trojúhelníku

$\mathbf{S_2}$ … souřadnice středu druhé strany trojúhelníku

$\mathbf{NV_1}$ … vektor osy první strany trojúhelníku

$\mathbf{NV_2}$ … vektor osy druhé strany trojúhelníku

$$S_1[x] \cdot NV_1[x] + S_1[y] \cdot NV_1[y] + C_1 = 0 \quad (6)$$

$$S_2[x] \cdot NV_2[x] + S_2[y] \cdot NV_2[y] + C_2 = 0 \quad (6)$$

Po dosazení zjistíme parametry C_1 a C_2:

$$C_1 = -S_1[x] \cdot NV_1[x] - S_1[y] \cdot NV_1[y]$$

$$C_2 = -S_2[x] \cdot NV_2[x] - S_2[y] \cdot NV_2[y]$$

Souřadnice průsečíku vektorů SV_1 a SV_2 se vypočítá následující rovnicí:

$$P[x] = \frac{\begin{vmatrix} NV_1[y] & C_1 \\ NV_2[y] & C_2 \end{vmatrix}}{\begin{vmatrix} NV_1[x] & NV_1[y] \\ NV_2[x] & NV_2[y] \end{vmatrix}}$$

(7)

$$P[y] = \frac{\begin{vmatrix} C_1 & NV_1[x] \\ C_2 & NV_2[x] \end{vmatrix}}{\begin{vmatrix} NV_1[x] & NV_1[y] \\ NV_2[x] & NV_2[y] \end{vmatrix}}$$

(7)

9.4.2. IMPLEMENTACE

Implementace všech senzorů je velmi jednoduchá. Jedná se pouze o to, že při kolizi vážky s některým senzorem dojde k zachycení této události v metodě *onSensorCollisionBegin*. V této metodě se provede zavolání funkce objektu vážky, která rozhodne, zda se vážka zastaví, začne zatáčet atd.

Nyní se pokusím sepsat scénáře, které mohou během létání uvnitř stromu nastat, a slovně popsat to, jak vážka na danou situaci zareaguje.

Scénář 1

- *Vážka je natočena na pozici „ODBOČ VLEVO / ODBOČ VPRAVO“.*
- *Přilétává k jakémukoliv senzoru ve větvi typu 2.*

Po kolizi se senzorem se načtou informace o tom, na které strany je možné zatočit. Poté se zkontroluje, zda již vážka neprovádí zatáčení. Pokud ne, provede se kontrola natočení vážky. Dojde k zahájení zatáčení na požadovaný úhel a přesun na koncový senzor křižovatky. Při střetu s tímto senzorem se opět provede kontrola, zda vážka zatáčí. V tomto případě zatáčení ukončí a vážka dále pokračuje v letu.

Scénář 2

- *Vážka je natočena na pozici „STOP“.*
- *Přilétává k jakémukoliv senzoru ve větvi typu 2.*

Po kolizi se senzorem se opět načtou informace o tom, na které strany je možné zatočit. Následně se také provede kontrola, zda již vážka nezatáčí. V dalším kroku se zjistí, že je vážka na pozici „*STOP*“, proto se nastaví rychlost vážky na 0 a nedovolí se jí pokračovat dále, dokud se nenatočí na pozici „*ODBOČ VLEVO* / *ODBOČ VPRAVO*“. Pokud se vážka otočí do protisměru, bude jí pohyb dopředu umožněn. Po opětovném natočení směrem ke křižovatce se provede kontrola, na kterou stranu je vážka natočena. V případě, že je natočena na pozici „*STOP*“, tak se opakuje celý scénář 2, jinak jí bude umožněno zatočení (viz scénář 1).

Scénář 3

- *Vážka je natočena na pozici „STOP“ nebo na pozici „ODBOČ VLEVO“ před první křižovatkou nebo na pozici „ODBOČ VPRAVO“ před druhou křižovatkou.*
- *Přilétává k prvnímu senzoru v první, resp. druhé křižovatce ve větvi typu 3.*

Při kolizi se senzorem se provedou stejné akce jako v minulých scénářích. Získají se informace o tom, kam je možné zatočit. Zkontroluje se, zda vážka právě nezatáčí. Pokud je vážka na pozici „*STOP*“, provede se kontrola, zda je možné křižovatkou projet rovně – to je možné pouze v křižovatkách ve větvi typu 3. Poté se neprovede automatické zatáčení, ale bude potřeba evidovat, že se vážka nachází mezi dvěma senzory – provádí „zatáčení“. Při kolizi s koncovým senzorem se zase nastaví, že došlo k ukončení zatáčení, a vážka může pokračovat dál.

Scénář 4

- *Vážka je natočena na pozici „STOP“.*
- *Přilétává k prvnímu senzoru v první, resp. druhé křižovatce ve větvi typu 3.*

Začátek scénáře je stejný jako ve scénáři 3. Rozdíl je v tom, že se může vážka na koncovém senzoru otočit do protisměru a vracet křižovatkou zpět. V tom případě se při otočení vážky na

senzoru musí opět evidovat, že dochází k „zatáčení“. Vážka v tomto případě projede křižovatkou opět rovně.

Scénář 5

- *Vážka je natočena na pozici „ODBOČ VLEVO“ před první křižovatkou nebo na pozici „ODBOČ VPRAVO“ před druhou křižovatkou.*
- *Přilétává k prvnímu senzoru v první, resp. druhé křižovatce ve větvi typu 3.*

Stejně jako v předchozím případě je začátek stejný jako ve scénáři 3. Nyní se po otočení do protisměru provede zahájení automatického zatáčení do levé, resp. pravé strany křižovatky.

Scénář 6

- *Vážka je natočena na pozici „STOP“ nebo na pozici „ODBOČ VLEVO“ před první křižovatkou nebo na pozici „ODBOČ VPRAVO“ před druhou křižovatkou.*
- *Přilétává k prvnímu senzoru v první, resp. druhé křižovatce ve větvi typu 3.*

Vážka se může otočit uprostřed křižovatky do protisměru a kolidovat tak opět s prvním senzorem křižovatky. V tom případě se stejně jako v předchozích případech ukončí „zatáčení“. Pokud se vážka otočí na tomto senzoru do protisměru, začnou se provádět kroky ze scénáře 3.

Scénář 7

- *Vážka je natočena na pozici „ODBOČ VPRAVO“ před první křižovatkou nebo na pozici „ODBOČ VLEVO“ před druhou křižovatkou.*
- *Přilétává k prvnímu senzoru v první, resp. druhé křižovatce ve větvi typu 3.*

Tento scénář je totožný se scénářem 1.

9.5. ZDROJE

K tomu, aby mohl uživatel získávat energii pro další růst stromu, je zapotřebí, aby se během pohybu uvnitř stromu snažil nasbírat co nejvíce objektů, které představují zdroje. O typech jednotlivých zdrojů jsem se zmínil již dříve. V této části se zaměřím na problém rozmísťování zdrojů. Každý zdroj se skládá celkem ze tří objektů:

- *3D MODEL ZDROJE*
- *HELPER*
- *SENZOR*

3D model zdroje

Jedná se vlastně o kouli, na kterou se následně mapuje materiál podle konkrétního typu zdroje. Každý zdroj může obsahovat různé množství energie. Velikost modelu zdroje bude závislá na energii, kterou zdroj obsahuje.

Helper

Helper je objekt, který je v ostré verzi hry neviditelný. V našem případě slouží k tomu, aby k němu byla namapována zvuková knihovna. Při sebrání zdroje se podle typu zdroje vybere zvuk z této knihovny, který se přehraje. Toto lze udělat také bez helperu, stačilo by namapovat zvuk přímo na 3D model zdroje. Problémem však bylo to, že při odstranění modelu zdroje sebraného vážkou, se přestane také přehrávat zvuk. Proto bylo třeba použít helper se zvukovou knihovnou. Jakmile uživatel sebere určitý zdroj, zavolá se metoda helperu, která spustí přehrávání zvuku. Samotný helper se odstraní až po přehrání tohoto zvuku.

Senzor

Senzor slouží opět k tomu, aby bylo možné zachytit kolizi s vážkou a zajistit tak správnou reakci na tuto situaci. Jeho velikost je o 25 % větší, než velikost 3D modelu. Je to z toho důvodu snadnějšího sbírání zdrojů uživatelem.

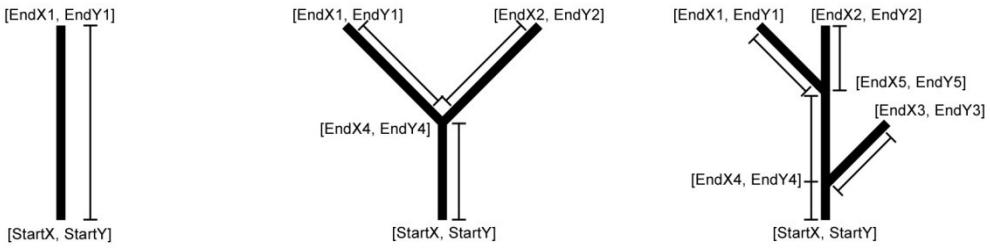
V následujícím návrhu proberu jak rozmístění jednotlivých zdrojů, tak i jejich funkci, která není u zdrojů nikterak složitá.

9.5.1. NÁVRH

Rozmístění jednotlivých zdrojů uvnitř větve bude náhodné. Náhodné však nebude to, kolik a jaký typ zdrojů bude ve větvi zobrazen. Toto záleží na tom, jakou energii získá větev z okolního světa.

Kromě zdrojů, které jsou získávány z okolního světa, existují také dva „zdroje“, které se generují náhodně v celém stromě. V jednom případě to je zdroj, který uživateli energii přidá a v druhém případě energii ubírá.

Zdroje jsou generovány zároveň při generování konkrétní větve. V případě získávání zdrojů z okolního světa nebereme větev jako celek, ale máme ji rozdělenou na několik částí. Podíváme-li se na obrázek obr. 15, zjistíme, že u větví typu 2 a 3 přibyly další souřadnice. Kromě souřadnic všech konců větví máme i souřadnice středů křižovatek. Větev se tedy skládá z pomyslných úseček, které lze na následujícím obrázku najít.



obr. 15 – náhled úseček

Při získávání energie z okolního světa tedy v konečné fázi nezískává energii jedna celá větev, ale pouze ta část větve, která se např. dotýká vody nebo zeminy v půdě, má na sobě umístěný list apod. Informace o tom, která větev má nasbíranou nějakou energii, jsou stejně jako informace o struktuře stromu uloženy v XML souboru. V dalším příkladu je ukázka toho, jak jsou tyto informace v XML uloženy.

```
<strokes>
      <stroke IDParent="1" IDStroke="4" MasterType="1" StartX="75"
              StartY="50" EndID="0" EndX="70" EndY="50" />

      <stroke IDParent="1" IDStroke="5" MasterType="1" StartX="75"
              StartY="54" EndID="1" EndX="70" EndY="60" />
</strokes>

<hashTree1Strokes*Point 4="13" 5="30">2</hashTree1Strokes*Point>
```

Pro každou větev typu 1–3 existuje v XML souboru seznam jednotlivých částí, ze kterých se větev skládá. Každá část má v rámci jednoho stromu jedinečné ID. Atribut *MasterType* určuje typ větve, které tato část patří. Atributy *StartX*, *StarY*, *EndX* a *EndY* určují počáteční a koncové souřadnice části. ID části je použito jako název atributu XML elementů pro ukládání informací o jednotlivých zdrojích.

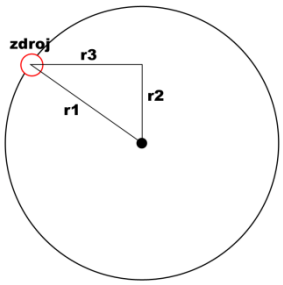
Při generování konkrétní větve dojde k nalezení všech částí, ze kterých se větev skládá. Nejprve se najdou části, které patří aktuální větvi. Poté se načtou hodnoty všech zdrojů, které každá část obsahuje. V následujícím kroku dojde ke generování 3D modelů zdrojů. Každý zdroj bude mít nastavenou maximální hodnotu energie, kterou může obsahovat. Navíc bude nastaveno omezení na maximální energii, kterou může obsahovat každá část větve.

Omezení jsou následující:

- *Každá část větve může obsahovat maximálně **100** jednotek každého druhu zdroje.*
- *Každý zdroj může obsahovat maximálně **50** jednotek energie.*

Z těchto omezení vyplývá, že v jedné části větve se najednou mohou zobrazit pouze dva 3D modely pro každý druh zdroje (nepočítají se zde tzv. „bonusové“ zdroje, které nejsou získávány z okolního světa).

Zdroje budou rozmísťovány náhodně téměř po 70 % délky každé části větve. Zdroje tedy nebudou umísťovány na prvních a posledních 15 % jednotlivých částí. To proto, aby se zamezilo umístění senzoru na místo křižovatky a na rozmezí dvou větví. Dále je potřeba, aby byly zdroje umístěny po obvodu větví. To se zajistí tak, že náhodně vygenerujeme pozici na ose konkrétní části větve. Nejprve je potřeba určit vertikální pozici zdroje. K tomu využijeme funkci *cosinus*. Jako parametr jí předáme náhodně vygenerované číslo z rozsahu $\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$. Výsledek funkce *cosinus* pak vynásobíme poloměrem větve. Posledním krokem byl výpočet horizontálního umístění zdroje. Zde využijeme Pythagorovu větu. Na následujícím obrázku (obr. 16) je znázorněn průřez větví. Strana r_1 znázorňuje poloměr větve. Stranu r_2 jsme vypočítali v předchozím kroku a nyní nám stačí vypočítat stranu r_3.



obr. 16 – umístění zdroje

Pro výpočet délky r_3 tedy použijeme následující vzorec:

$$r_3 = \sqrt{r_1^2 - r_2^2} (8)$$

V této chvíli by byly všechny zdroje umístěny vždy na jedné straně. Proto délku strany r_3 ještě vynásobíme náhodně vybraným číslem z množiny {1,-1}.

To bylo vše, co se týká rozmísťování zdrojů. Nyní se také krátce zmíním o tom, co se stane, když uživatel sebere některý ze zdrojů. Tato část je vcelku triviální, proto se jí již nebudu věnovat v samostatné kapitole.

Po kolizi vážky se senzorem zdroje se provedou následující kroky:

- *3D model zdroje se odstraní ze scény.*
- *Přehraje se zvuk, který signalizuje sebrání zdroje.*
- *Hodnota zdroje je přičtena/odečtena k/od celkové energie hráčova stromu.*
- *Hodnota zdroje je odečtena z energie časti větve, ve které se zdroj nacházel.*
- *Dojde k odstranění helperu, na který je namapována knihovna zvuků.*

9.5.2. IMPLEMENTACE

V této části se budu zabývat pouze implementací rozmístění senzorů, což je více zajímavé než implementace toho, jak bude aplikace reagovat na sebrání některého zdroje vážkou.

```
Set nResourcesCount to nCollectedEnergy / nEnergyPerResource
Set i to 0
While i < nResourcesCount do
      Set ResourcePosition to (nStrokeStartX, nStrokeStartY) +
            Random(0.15, 0.85) * Length(currentStroke)

      Set nResourceX to cos(Random(-180, 180)) * nStrokeRadius
      Set nResourceY to sqrt(pow(nStrokeRadius, 2) -
            pow(nResourceX, 2)) * Random({1,-1})

      Set ResourcePosition[X] to ResourcePosition[X] + nResourceX
      Set ResourcePosition[Y] to ResourcePosition[Y] + nResourceY

      i++
Endwhile
```

Nejprve se tedy vypočítá, kolik zdrojů je možné vytvořit. Poté se pro každý zdroj vygeneruje náhodná pozice. K souřadnicím začátku části větve se přičte vektor, který má náhodnou velikost z rozsahu 15–85 % z velikosti části větve. V dalších krocích se postupně vypočtou strany r_2 a r_3. Nakonec se tyto výsledky přičtou k pozici, která byla vygenerována na začátku cyklu.

9.6. POHYB VÁŽKY

Následující dvě kapitoly se budou zaobírat částmi hry, které nejvíce ovlivní hratelnost. Jedná se o pohyb vážky a následně její ovládání. Těmto částem jsme se proto věnovali co nejvíce a snažili jsme se tedy vyzkoušet velké množství možností, jak je řešit. Nejprve proberu pohyb vážky.

Uvažovali jsme nad dvěma typy pohybů. Jeden, který dával uživateli naprostou volnost. Uživatel tak mohl natáčet vážku všemi směry a létat po celém prostoru vytvořeného stromu. Druhý způsob, který je nakonec použit ve finální verzi, nedává uživateli takovou svobodu. Vážce je umožněn pouze pohyb po obvodu větve, let kupředu a otočení o 180°. Zatáčení je pak řešeno pomocí mechanismu, který byl popsán výše. Oba způsoby jsme nakonec implementovali, takže se nyní pokusím popsat výhody/nevýhody každého z nich.

9.6.1. VOLNÝ POHYB

Jednoznačnou výhodou tohoto pohybu je, že je uživatel omezován pouze stěnou stromu, jinak má naprostou volnost. U tohoto pohybu se také velmi nabízelo využití fyzikálního modelu pro pohyb přímo od tvůrců Shivy. Před tím, než jsme se rozhodli, že jako objekt, který bude uvnitř stromu létat, použijeme model vážky, uvažovali jsme o použití modelu létající lodě. Toto nás

tedy přivedlo na myšlenku použití právě již hotového fyzikálního modelu od Shivy. Tento model nám poskytoval velmi mnoho nástrojů k tomu, aby byl pohyb lodi co nejvíce přiblížen reálnému pohybu letadel. Ve výsledku tedy stačilo nastavit základní fyzikální vlastnosti jako váhu, tření, pružnost a v neposlední řadě také rychlost. Pro určení směru se pak používají úhly ve třírozměrném prostoru. Tyto tři úhly jsou známy z letectví pod pojmy *roll*, *pitch* a *yaw*. Podle zadaných parametrů a nastavených úhlů pak fyzikální model sám vypočítal trasu objektu tak, aby co nejvíce odpovídal reálnému světu. Mezi další výhody tohoto modelu patřilo také to, že obsahoval tzv. „kolizní systém“, díky kterému nemůže řízený objekt proletět objekty, které byly označeny jako pevný objekt.

To všechno zatím byly výhody při použití fyzikálního modelu. Později se ale ukázalo, že není možné tento fyzikální model použít kvůli jedné překážce. Tou bylo ovládání takto implementovaného pohybu. Většina dnešních simulátorů, které používají fyzikální modely, umožňují uživateli pohyb v relativně volném prostoru. Nehrozí tak, že by řízený objekt každou chvíli narážel do okolních předmětů. V našem případě je pohyb řízeného předmětu omezen ze všech stran modelem stromu. Z toho důvodu docházelo při tomto pohybu každou chvíli ke kolizi se stěnou stromu, což hráči značně ztěžovalo pohyb. Toto byl tedy hlavní důvod k tomu, proč použít pohyb, který je pro hráče více omezující.

9.6.2. OMEZENÝ POHYB

Omezený pohyb má oproti volnému pohybu docela striktně stanovený směr, kterým se vážka smí pohybovat. Uživatel může kontrolovat rychlost a pozici vážky na obvodu každé větve. Nemusí se tak starat o přesné řízení vážky uprostřed větve a může se více soustředit na sbírání jednotlivých zdrojů. V tomto způsobu tedy odpadá řešení kolizí se stěnou stromu. Pro otočení do protisměru musela být implementována metoda, která postupně otočí model vážky o 180°, a až poté může uživatel pokračovat dále v cestě.

Jako nevýhodu můžeme brát, že nebylo možné použít již hotový fyzikální model, ale bylo třeba implementovat vlastní, který při každém obnovení obrazovky nastaví novou pozici vážky v závislosti na nastavené rychlosti a směru, kterým je vážka natočená.

9.6.3. NÁVRH

Z předchozího textu vyplývá, že jsme ve finální verzi použili omezený pohyb. Proto se v návrhu budu zabývat právě jím.

Shiva nabízí metody, které jsou volány při každém obnovení obrazovky. Pro náš pohyb tedy tyto metody využijeme. Samotná vážka se skládá ze tří objektů – model vážky, senzor na modelu vážky a senzor, který slouží ke kolizi se senzory na křižovatkách. Uprostřed tohoto posledního senzoru je umístěn pivot, kolem kterého se vážka otáčí. Tento pivot bude po načtení hry umístěn na středové ose první větve. Při obnovení obrazovky se provedou kroky v závislosti

na dané situaci. Buď se jedná o obyčejný pohyb ve směru, na který je vážka natočena nebo může probíhat zatáčení na křižovatce či otáčení do protisměru.

Jak již bylo řečeno, pozice vážky se mění při každém obnovení obrazovky. Délka intervalu, kdy dochází k překreslení obrazovky, není konstantní. Někdy může jedno překreslení trvat 0,01 vteřinu, jindy např. 0,9. Proto může docházet k tomu, že se bude uživateli zdát pohyb vážky kolísavý. Z toho důvodu musí byt pohyb závislý na tom, jak dlouhou dobu intervaly mezi překreslením obrazovky průměrně trvají. Nyní se pokusím popsat to, co se s vážkou děje v každé ze tří zmíněných situací.

9.6.3.1. POHYB VE SMĚRU VÁŽKY

Obsluha této situace byla velmi jednoduchá. Nejprve se zkontroluje, zda je vůbec pohyb dopředu umožněn (vážka nestojí na konci stromu nebo před křižovatkou). Pokud ano, tak je možné ihned provést změnu pozice vážky. Shiva umožňuje změnit pozici relativně k aktuální pozici vážky a také ve směru, kterým je model vážky natočen. V našem případě tedy budeme vážku posouvat o hodnotu z následujícího vzorce:

$$NewPosition[x] = OldPosition[x] + Speed \cdot AvgFrameTime$$

Vážku posunujeme pouze po x-ové souřadnici lokálního souřadnicového systému, který je určen natočením vážky. *Speed* představuje aktuálně nastavenou rychlost vážky a *AvgFrameTime* zase průměrnou dobu, za kterou dojde k překreslení obrazovky.

9.6.3.2. ZATÁČENÍ NA KŘIŽOVATCE

Při zatáčení na křižovatkách využíváme hodnoty parametrů, které jsme získali při kolizi s prvním senzorem křižovatky. Pro pohyb budeme potřebovat přepočítat lineární rychlost vážky na úhlovou. Pro tento výpočet použijeme následující rovnici:

$$AngularSpeed = \frac{LinearSpeed^2}{Radius} \tag{9}$$

AngularSpeed představuje úhlovou rychlost v radiánech/s. *LinearSpeed* je lineární rychlost vážky, která je již vynásobena průměrnou dobou intervalu překreslení obrazovky, a *Radius* představuje poloměr kružnice, kterou bude vážka opisovat. Tento poloměr se vypočítá z hodnot, které jsme získali z prvního senzoru. Jedná se o souřadnici prvního senzoru a souřadnici středu pro zatáčení. Po odečtení těchto bodů od sebe získáme vektor, jehož velikost představuje poloměr kružnice, kolem které se bude zatáčení provádět. Posledním krokem k vypočtení rychlosti, kterou se bude vážka při průletu křižovatkou pohybovat, je převedení úhlové rychlosti z radiánů/s na jednotky z prostředí Shivy. K tomu se použije následující rovnice:

$$Angle = \frac{RadAngle \cdot 180}{\pi}$$

(9)

Shiva nabízí metody pro pohyb objektu po kružnici. Stačí znát pouze střed kružnice a velikost úhlu, o který se má objekt otočit. Střed jsme získali z prvního senzoru křižovatky a velikost úhlu zjistíme z předchozích rovnic. Při každém obnovení obrazovky se tak vážka o tento úhel natočí. Z prvního senzoru křižovatky jsme také získali úhel, o který se má vážka natočit. Při zatáčení vážky tak zkontrolujeme, zda se vážka již o tento úhel neotočila. Pokud ano, tak se zatáčení ukončí a vážka se přemístí na souřadnice koncového senzoru křižovatky. To se provede jen kvůli tomu, abychom si byli jisti, že je po zatáčení vážka opět umístěna na ose větve.

9.6.3.3. OTOČENÍ DO PROTISMĚRU

Poslední situací je otočení vážky o 180°. Řešení je v základu velmi podobné jako zatáčení na křižovatkách. Po vyvolání této akce se přeruší pohyb ve směru natočení vážky a začne se provádět otáčení okolo osy vážky. Toto otáčení není závislé na aktuální rychlosti vážky. Proto se při každém obnovení obrazovky provede otočení o 5°. Zatáčení bude přerušeno, jakmile se vážka otočí o celých 180°.

9.6.4. IMPLEMENTACE

Stejně jako v návrhu, tak i nyní rozdělím implementaci na části, ve kterých postupně ukáži, jak vypadá implementace pohybu dopředu, zatáčení na křižovatkách a otočení do protisměru.

9.6.4.1. POHYB VE SMĚRU VÁŽKY

Následující příklad zobrazuje, jakým způsobem je řešen pohyb ve směru vážky.

```
If bCanMove then
      Set nMoveLength to nCurrentSpeed * nAvgFrameTime
      Move in X-axe with nMoveLength offset
Endif
```

Pokud je aktuální rychlost rovná nule, tak také hodnota proměnné *MoveLength* bude nulová a vážka se nepohne. V opačném případě dojde k posunu vážky ve směru osy X.

9.6.4.2. ZATÁČENÍ NA KŘIŽOVATCE

Ze všech tří pohybů je tento na implementaci nejnáročnější. Následující kroky jsou provedeny poté, co vážka koliduje s prvním ze senzorů, které jsou umístěny na křižovatce.

```
If bCanTurn then
      Set nCurrentAngle to 0
```

```
      For each frame do
            If nCurrentAngle < nNextAngle then
            Set nRadisu to length of ([nSensorX,nSensorY] -
                  [nTurnCenterX,nTurnCenterY])
            Set nAngularSpeed to (nCurrentSpeed * nAvgFrameTime)^2 /
                  nRadius
            Set nAngle to (nAngularSpeed * 180) / 3.14

            If nCurrentAngle + nAngle > nNextAngle then
                  Set nAngle to nNextAngle - nCurrentAngle
            Endif

            Rotate around [nTurnCenterX,nTurnCenterY] with angle nAngle
            Set nCurrentAngle to nCurrentAngle + nAngle
      Endfor
Endif
```

9.6.4.3. OTOČENÍ DO PROTISMĚRU

Poslední ukázkou bude implementace otočení vážky do protisměru. Jakmile se uživatel rozhodne k otočení zpět o 180°, provede se následující kód:

```
Set nCurrentAngle to 0

For each frame do
      If nCurrentAngle < 180 then
            Rotate with angle 5
            Set nCurrentAngle to nCurrentAngle + 5
      Endif
Endfor
```

9.7. OVLÁDÁNÍ

Stejně jako při výběru typu pohybu, rozhodovali jsme se i u typu ovládání mezi dvěma variantami. Díky tomu, že všechna zařízení, na která jsme hru cílili, měla k dispozici gyroskop, rozhodli jsme se pro využití možností, které tento nástroj nabízí. Nejprve jsme si představovali, že budeme pohyb vážky ovládat pomocí gyroskopu, a to ve všech směrech. Dalším typem bylo z části použití gyroskopu a z části námi implementovaných ovládacích prvků na obrazovce. Nakonec jsme také vyzkoušeli ovládání čistě pomocí ovládacích prvků na obrazovce.

9.7.1. OVLÁDÁNÍ GYROSKOPEM

První, co nás napadlo, bylo použít pro ovládání vážky pouze gyroskop, který obsahuje každé z námi vybraných zařízení. Při naklánění zařízení směrem od sebe a k sobě mohl uživatel nastavovat rychlost, kterou se bude vážka pohybovat. Natáčením doleva a doprava zase polohu vážky na obvodu větve. Pokud by se chtěl uživatel otočit do protisměru, musel by držet zařízení nakloněné k sobě tak dlouho, dokud by vážka úplně nezastavila, a až po chvíli by došlo k samotnému otočení. Od této varianty jsme velmi rychle upustili. Problém nebyl jen v otáčení

do protisměru, ale také v otáčení po stěně větve. Velmi dlouhou dobu trvalo, než se uživatel natočil tak, jak by chtěl. Toto mu velice znepříjemňovalo sběr zdrojů. Rychlost zatáčení se zde dala upravovat podle toho, jak moc bylo zařízení natočeno na levou či pravou stranu. Čím více bylo natočeno, tím rychleji se vážka otáčela. Uživatel byl nucen hlídat si polohu zařízení ve všech směrech, což pro něj bylo velmi obtížné.

9.7.2. OVLÁDÁNÍ GYROSKOPEM A JOYPADEM

Z důvodu, že si uživatel musel hlídat polohu zařízení ve všech směrech, jsme se rozhodli, že sice u použití gyroskopu zůstaneme, ale nebudeme jej využívat pro všechny typy pohybů. Gyroskop jsme tedy používali pro ovládání natočení vážky na obvodu stěny, ale pro nastavení rychlosti jsme použili ovládací prvky na obrazovce. Toto řešení se zdálo být přijatelnější než přechozí. Stále zde ale přetrvával problém s tím, že nebylo možné během chvíle vážku přesunout z jedné strany větve na druhou. Sbírání zdrojů, a vlastně také zatáčení na křižovatkách, bylo stále velmi obtížné. Proto jsme ani tento způsob ovládání ve finální verzi nepoužili.

9.7.3. OVLÁDÁNÍ DVĚMA JOYPADY

Posledním způsobem je ovládání čistě pomocí ovládacích prvků, které jsme umístili na obrazovku. Jeden je umístěn v levém spodním rohu obrazovky a slouží k určení pozice vážky na stěně větve. Další je umístěn na pravé straně obrazovky a slouží k nastavení rychlosti, kterou vážka poletí. Posledním prvkem je tlačítko, po jehož stisknutí se ihned provede otočení do protisměru. Zvolení tohoto způsobu se nakonec ukázalo jako velmi vhodné. Uživatel určí na levém prvku pouhým umístěním prstu místo, na kterém se má vážka nacházet. Jedná se vlastně o náhled průřezu větve a uživatel se dotkne tam, kam chce vážku přemístit. Vážka se na toto místo přesune ihned.

Také ovládání druhého ovládacího prvku je velmi snadné. Tento prvek je implementován formou vertikálního posuvníku. Čím je posuvník výše, tím rychleji vážka letí.

9.7.4. NÁVRH

Také u návrhu ovládání se budu zabývat pouze tím způsobem, který jsme nakonec implementovali ve finální verzi hry. Nejprve popíši způsob rozmístění ovládacích prvků a následně to, jak fungují.

9.7.4.1. ROZMÍSTĚNÍ NA OBRAZOVCE

V předchozím textu jsem se již zmínil o tom, jak jsou jednotlivé prvky umístěny na obrazovce. Nyní se to pokusím více upřesnit. Shiva používá pro určení pozice nebo rozměrů jednotlivých prvků, jako jsou tlačítka, panely, texty aj., procenta z celkové velikosti obrazovky. Tento systém je na jednu stranu pochopitelný – aplikace vyvíjené Shivou jsou cíleny na velké množství

zařízení, které mají velmi rozdílné rozlišení. Proto se mohlo stát to, že by tlačítko, které má přijatelnou velikost na zařízení s nízkým rozlišením, bylo na zařízení s vysokým rozlišením skoro neviditelné. Na druhou stranu to přináší problém s tím, že dochází k deformaci těchto prvků na zařízeních s rozdílným poměrem stran. Kvůli tomu mohou být textury použité na pozadí těchto prvků ve špatné kvalitě. Z těchto důvodů je potřeba upravit velikost těchto ovládacích prvků pro konkrétní zařízení až za běhu aplikace. U jopadů jsme vyžadovali, aby byla jejich velikost stejná na všech rozlišeních.

Výpočet velikosti bude pro všechny ovládací prvky totožný. U každého joypadu známe originální velikost textury, která má být na joypad použita. Musíme tedy zjistit, kolik procent bude každý joypad na cílovém zařízení zabírat. K tomu potřebujeme znát rozlišení obrazovky zařízení a rozlišení textury. Podle následujících rovnic zjistíme šířku a následně výšku textury v procentech:

$$WidthInPer = \frac{TextureWidth}{ScreenWidth} \cdot 100$$

$$HeightInPer = \frac{TextureHeight}{ScreenHeight} \cdot 100$$

9.7.4.2. FUNKCE JOYPADŮ

V následující části se pokusím popsat, co nastane, když se uživatel dotkne některého z joypadů. Omezím se pouze na joypad, který ovládá natočení vážky na stěně větve a joypadu pro ovládání rychlosti vážky. Funkcí tlačítka pro otočení do protisměru se zde zabývat nebudu. Řešení tohoto problému je triviální.

Stejně jako jsme u senzorů pracovali s metodou *onSensorCollision*, budeme u ovládacích prvků na obrazovce pracovat s metodami *onTouchSequence*. Shiva nabízí tři handlery, které zachytávají události, jež nastanou při dotyku uživatele na displej zařízení:

- *onTouchSequenceBegin*
- *onTouchSequenceChange*
- *onTouchSequenceEnd*

onTouchSequenceBegin – handler zachytí situaci, kdy uživatel položí některý prst na displej zařízení.

onTouchSequenceChange – jak již název napovídá, tento handler se zavolá, když uživatel změní polohu prstu na displeji. Tato metoda je volána s parametry, v nichž jsou uloženy souřadnice, na kterých se uživatel displeje dotýká. Najednou lze kontrolovat až tři dotyky uživatele.

onTouchSequenceEnd – poslední handler je zavolán, když uživatel uvolní prst z displeje.

U všech joypadů známe souřadnice levého spodního a pravého horního rohu. Proto bude velmi snadné zjistit, zda se uživatel dotkl displeje na místě, na kterém je umístěn joypad. Stačí zkontrolovat, zda je souřadnice X větší, než X-ová souřadnice levého spodního rohu joypadu a zároveň nižší, než X-ová souřadnice pravého horního roku. Toto byla první část ověření. Nyní ještě potřebujeme provést naprosto stejné kroky pro souřadnici Y.

Levý joypad má kruhový tvar. Při dotyku na tento joypad se bere střed tohoto kruhu jako počátek souřadnic. Nyní vytvoříme vektor *u = (0, -1)*, který slouží pouze jako svislá osa souřadnic, od které budeme chtít zjistit úhel, o který je posunut druhý vektor *v*. Ten vede od středu joypadu po místo, kterého se uživatel dotknul. Ke zjištění úhlu, který tyto dva vektory svírají, použijeme výpočet skalárního součinu:

$$\alpha = \cos^{-1}\left(\frac{u_1 v_1 + u_2 v_2}{\sqrt{u_1^2 + u_2^2} \cdot \sqrt{v_1^2 + v_2^2}}\right)$$

(10)

Tento úhel se pak použije k tomu, aby se změnila pozice vážky vzhledem ke svislé ose větve. Na obrázku obr. 17 je přesněji vidět to, jak vypadají oba vektory.

obr. 17 – levý joypad

Pravý joypad, který ovládá rychlost vážky, obsahuje také prvek, který mění svou polohu podle toho, na které místo uživatel přesunul prst. Pohyb tohoto prvku je omezen pouze na osu Y. Uživatel jej tak může přesunovat pouze nahoru nebo dolů. Pokud zjistíme, že uživatel položil prst na místo, kde se tento joypad nachází, přepočítáme souřadnice relativně od středu spodní strany joypadu. Pokud je tedy tento střed umístěn na souřadnicích *JStartX*, *JStartY* a uživatel se dotkne displeje na místě *UTouchPointX*, *UTouchPointY*, spočítá se výsledná souřadnice takto:

$$RelX = UTouchPointX - JStartX$$

$$RelY = UTouchPointY - JStartY$$

9.7.5. IMPLEMENTACE

Jako ukázku implementace joypadů jsem vybral to, jakým způsobem se kontroluje dotyk uživatele na místo, kde se nachází joypad, a následný přepočet absolutních souřadnic na relativní od středu joypadu. Nakonec také provádím výpočet úhlu, na který se má vážka natočit vzhledem ke svislé ose větve.

```
If nTouchPointX > nJoypadLeftBottomX AND
   nTouchPointX < nJoypadRightTopX   AND
   nTouchPointY > nJoypadLeftBottomY AND
   nTouchPointY < nJoypadRightTopY then
      Set nRelativeX to nTouchPointX - nJoypadCenterX
      Set nRelativeY to nTouchPointY - nJoypadCenterY

      Set nBaseVectX to 0
      Set nBaseVecY to -1

      Set nAngle to arcos((nBaseVectX*nRelativeX +
            nBaseVectY * nRelativeY) /
            sqrt(pow(nBaseVectX, 2) + pow(nBaseVectY, 2)) *
            sqrt(pow(nRelativeX, 2) + pow(nRelativeY, 2))
Endif
```

Nejprve se zkontroluje, zda se uživatel dotkl displeje na místě, kde je umístěný joypad. Pokud ano, tak se provede výpočet relativních souřadnic od středu joypadu. Následně se nastaví hodnoty vektoru, který představuje svislou osu větve. V posledním kroku se vypočítá úhel, který svírá svislá osa větve s vektorem, který vede od středu souřadnic po souřadnice, na kterých se uživatel dotkl displeje.

10. NASAZENÍ NA CÍLOVOU PLATFORMU

V této poslední kapitole se stručně zaměřím na to, jak se dá pomocí nástrojů společnosti Stonetrip vygenerovat zdrojový kód, ze kterého je možné následně sestavit spustitelnou aplikaci pro konkrétní cílovou platformu. Ve své práci jsme se zaměřovali především na operační systémy iOS a Android OS, nyní popíši, jak vygenerovat zdrojový kód, který je napsán v jazyce Objective-C, resp. Java.

K převodu zdrojových kódů, které jsou napsány v jazyce Lua, vyvinul Stonetrip nástroj nazvaný Shiva Authoring Tool, který lze spustit na operačních systémech Windows a MacOS. Tento nástroj prochází kódy v jazyce Lua a podle vnitřních pravidel jej transformuje do zdrojových kódů napsaných v programovacím jazyku, který je nativní pro konkrétní cílovou platformu. Při exportování do zařízení s iOS a Android OS je také potřeba, aby měl uživatel nainstalovány SDK knihovny, které jsou pro vývoj určeny.

10.1. iOS

Nejprve popíši to, co je vše potřeba k vytvoření spustitelného instalačního balíku pro iOS. Před začátkem generování je nutné, aby měl uživatel nainstalován iOS SDK a vývojové prostředí XCode. Z toho vzniká určité omezení na operační systém, na kterém je možné balík pro iOS vytvořit. iOS SDK je totiž k dispozici pouze pro operační systémy MacOS.

Po přidání zdrojových kódů do aplikace Shiva Authoring Tool je uživatel vyzván, aby zadal cestu k obrázku, který bude použit jako ikona aplikace na cílovém zařízení. Dále je možné uvést cestu k obrázku, který se uživateli zobrazí před tím, než bude aplikace úplně načtena v paměti zařízení. Dalším krokem je ověření aplikace. To je pro export do iOS specifické. Je zde totiž vyžadováno, aby měl uživatel na svém počítači nahrán tzv. „provision profile“, který je součástí vývojové licence od Apple. Bez tohoto podpisu by nebylo možné aplikaci v koncovém zařízení spustit.

Po provedení všech kroků je již vše v režii Authoring Toolu. Z námi vložených zdrojových kódu vytvoří kód v Objective-C, ze kterého vznikne projekt vývojového prostředí XCode. Nyní již není poznat, že byla aplikace vytvořena pomocí Shiva 3D editoru, a vypadá, jako by byla od základů napsána v Objective-C.

10.2. ANDROID OS

Vytváření instalačního balíku pro Android OS je velmi podobné tomu pro iOS. Rozdíly jsou hlavně v požadavcích před samotným spuštěním Authoring Toolu. Stejně jako v případě iOS bylo potřeba nainstalovat Android SDK. Jeho použití není vázáno na určitý operační systém, proto je možné vytvořit instalační balík pro iOS všude tam, kde je možno nainstalovat Shiva Authoring Tool. Dále je potřeba, aby měl uživatel nainstalován Android NDK. Při použití operačního systému Windows je také nutné mít k dispozici aplikaci Cygwin, která umožňuje

používat nástroje, které jsou základem operačního systému Linux. Při použití OS Windows je potřeba mít nainstalováno vývojové prostředí Eclipse, které pro svůj běh využívá Java SE Develompent Kit, proto je nutné mít také tyto knihovny na svém PC. Konečně posledním požadavkem je ADT plugin pro Eclipse. Tento plugin umožňuje vyvíjet aplikace pro Android právě v prostředí Eclipse. Obsahuje např. nástroj pro vytváření uživatelského rozhraní, simulátory koncových zařízení a také nástroj pro vytváření instalačních balíku *.apk*.

Shiva 3D umí namísto souboru STK vygenerovat zdrojové kódy v jazyce C++. Toto se využívá právě u exportu do Android OS. Následující část je totožná s přidáním aplikace do Authoring Toolu u iOS. Jediný rozdíl je v tom, že není třeba vyplňovat údaje o „provision profilu“.

Po skončení generování zdrojových kódů aplikací Authoring tool je vytvořen projekt pro prostředí Eclipse, který obsahuje aplikaci se zdrojovými kódy v jazyce Java.

11. ZÁVĚR

Cílem diplomové práce byl vývoj herní aplikace pro zařízení s iOS a Android OS, na které jsme pracovali v týmu ve společnosti Zoongo, s. r. o. Prošli jsme všemi částmi vývojového cyklu. Od specifikací požadavků přes návrh a implementaci, až po provádění základních testů a následnou distribuci aplikace do on-line obchodů.

V rámci této práce nejprve uvádím základní teoretické znalosti, které by měly být dostačující k pochopení toho, s jakými nástroji jsme během vývoje pracovali. Ať už se jedná o popsání vývojového prostředí Shiva 3D nebo popis jazyka UML aj.

Většina ukázek implementace se zabývá tou částí hry, o kterou jsem se převážně staral já – 3D svět. Mohl jsem si tak vyzkoušet, jakým způsobem se pracuje s 3D objekty, které byly vytvořeny v některém z grafických nástrojů, ale také s jednoduchými tvary, které lze vytvářet přímo v editoru Shiva 3D. Podstatnou částí také bylo vytvoření jednoduchého fyzikálního modelu pro hlavní „postavu“ celé hry – vážku. Zde jsem řešil mnoho problémů pomocí prvků z analytické geometrie. Mým dalším úkolem v týmu byla správa serveru, který jsme k vývoji používali a na kterém jsme měli spuštěny služby jako SVN, Samba, Bugzilla.

V současné době je hra distribuována pomocí on-line obchodu společnosti Apple – App Store. Zde nabízím odkaz přímo na stránky apple.com:

http://itunes.apple.com/gb/app/re!sources/id508545253 (odkaz z 19. 4. 2012)

Následujícím krokem bude umístění hry do obchodu Google Play, který ještě donedávna nepovoloval vkládání komerčních aplikací z území České republiky. Od 12. 4. 2012 však byla Česká republika přidána k zemím, ze kterých je možné distribuovat placené aplikace přes Google Play[14].(11)

[14] https://play.google.com/store?hl=en

POUŽITÁ LITERATURA

1. **FIGUEIREDO, L. H. DE and L. H.W.** *Lua Programming Gems.* Rio de Janeiro : Lua.org, 2008.

2. **ARLOW, Jim a NEUSTADT, Ila.** *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky.* místo neznámé : Computer Press, 2007. ISBN 978-80-251-1503-9.

3. **PILATO, C a NEUSTADT, Ila.** *Version control with subversion: objektově orientovaná analýza a návrh prakticky.* Beijing : O'Reilly, 2008. ISBN 978-0-596-51033-6.

4. Počítačová hra. *Wikipedia.org.* [Online] 21. 3 2012. [Citace: 17. 4 2012.] http://cs.wikipedia.org/wiki/Po%C4%8D%C3%ADta%C4%8Dov%C3%A1_hra#.C5.BD.C3.A1nry.

5. **KONČEL, Jan.** Velikost vektoru. *Analytická geometrie.* [Online] 2009. [Citace: 18. 4 2012.] http://www.karlin.mff.cuni.cz/katedry/kdm/diplomky/jan_koncel/vektory.php?kapitola=velikost.

6. **VOJÁČEK, Jakub.** Matematika pro každého. *Analytická geometrie - Obecná rovnice přímky.* [Online] 24. 5 2008. [Citace: 18. 4 2012.] http://maths.cz/clanky/analyticka-geometrie-obecna-rovnice-primky.html.

7. **DANNY, D.** Wikibooks.org. *Vzájemná poloha dvou přímek.* [Online] 28. 9 2010. [Citace: 18. 4 2012.] http://cs.wikibooks.org/wiki/Vz%C3%A1jemn%C3%A1_poloha_dvou_p%C5%99%C3%ADmek.

8. **WEISSTEIN, Eric W.** Pythagorean Theorem. *MathWorld.* [Online] 12. 4 2012. [Citace: 18. 4 2012.] http://mathworld.wolfram.com/PythagoreanTheorem.html.

9. **FINC, E. Joseph.** *Physics for Athletes and Other Serious Students.* místo neznámé : Mohican Pub, 1988. ISBN 978-0923231101.

10. **WEISSTEIN, Eric W.** Dot Product. *MathWorld.* [Online] 12. 4 2012. [Citace: 18. 4 2012.] http://mathworld.wolfram.com/DotProduct.html.

11. **HRÁČEK, Filip.** Čeští vývojáři teď mohou prodávat aplikace pro Android. *Google blog ČR.* [Online] Google ČR, 12. 4 2012. [Citace: 19. 4 2012.] http://google-cz.blogspot.com/2012/04/cesti-vyvojari-ted-mohou-prodavat.html?spref=tw.